APPLICATIONS OF PARALLEL GLOBAL OPTIMIZATION TO MECHANICS PROBLEMS

By

JACO FRANCOIS SCHUTTE

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2005
Copyright 2005

by

JACO FRANCOIS SCHUTTE
This work is dedicated to my parents and my wife Lisa.
ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr. Raphael T. Haftka, chairman of my advisory committee, for the opportunity he provided me to complete my doctoral studies under his exceptional guidance. Without his unending patience, constant encouragement, guidance and expertise, this work would not have been possible. Dr. Haftka's mentoring has made a lasting impression on both my academic and personal life.

I would also like to thank the members of my advisory committee, Dr. Benjamin Fregly, Dr. Alan D. George, Dr. Panos M. Pardalos, and Dr. Nam Ho Kim. I am grateful for their willingness to serve on my committee, for the help they provided, for their involvement with my oral examination, and for reviewing this dissertation. Special thanks go to Dr. Benjamin Fregly, who provided a major part of the financial support for my studies. Special thanks also go to Dr. Alan George whose parallel processing graduate course provided much of the inspiration for the research presented in this manuscript, and for reviewing some of my publications. Thanks go also to Dr. Nielen Stander who provided me with the wonderful opportunity to do an internship at the Livermore Software Technology Corporation.

My colleagues in the Structural and Multidisciplinary Optimization Research Group at the University of Florida also deserve many thanks for their support and the many fruitful discussions. Special thanks go to Tushar Goel, Erdem Acar, and also Dr. Satchi Venkataraman and his wife Beth, who took me in on my arrival in the USA and provided me a foothold for which I will forever be grateful.
The financial support provided by AFOSR grant F49620-09-1-0070 to R.T.H. and the NASA Cooperative Agreement NCC3-994, the "Institute for Future Space Transport" University Research, Engineering and Technology Institute is gratefully acknowledged.

I would also like to express my deepest appreciation to my parents. Their limitless love, support and understanding are the mainstay of my achievements in life.

Lastly, I would like to thank my wife, Lisa. Without her love, patience and sacrifice I would never have been able to finish this dissertation.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>x</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xiii</td>
</tr>
</tbody>
</table>

## CHAPTER

1 INTRODUCTION ...........................................................................................................1

   Statement of Problem ..........................................................................................1
   Purpose of Research ...........................................................................................1
   Significance of Research
       Parallelism by Exploitation of Optimization Algorithm Structure ..............2
       Parallelism through Multiple Independent Concurrent Optimizations ...........3
       Parallelism through Concurrent Optimization of Decomposed Problems ........3
   Roadmap ..............................................................................................................4

2 BACKGROUND .........................................................................................................5

   Population-based Global Optimization...........................................................5
   Parallel Processing in Optimization ..................................................................6
   Decomposition in Large Scale Optimization .....................................................7
   Literature Review: Problem Decomposition Strategies .....................................8
       Collaborative Optimization (CO) ..................................................................8
       Concurrent SubSpace Optimization (CSSO) ...................................................11
       Analytical Target Cascading (ATC) ...............................................................15
       Quasiparallel Decomposition and Optimization ..........................................17

3 GLOBAL OPTIMIZATION THROUGH THE PARTICLE SWARM ALGORITHM ..................18

   Overview ............................................................................................................18
   Introduction .......................................................................................................19
   Theory ...............................................................................................................21
   Particle Swarm Algorithm ..................................................................................21
4 PARALLELISM BY EXPLOITING POPULATION-BASED ALGORITHM STRUCTURES

Overview .................................................................................................................47
Introduction ..............................................................................................................48
Serial Particle Swarm Algorithm .............................................................................50
Parallel Particle Swarm Algorithm ........................................................................53
  Concurrent Operation and Scalability ...............................................................53
  Asynchronous vs. Synchronous Implementation ..............................................54
  Coherence .........................................................................................................55
  Network Communication ..................................................................................56
  Synchronization and Implementation .............................................................58
Sample Optimization Problems ........................................................................59
  Analytical Test Problems ...............................................................................59
  Biomechanical System Identification problems ..........................................60
  Speedup and Parallel Efficiency ....................................................................63
Numerical Results .....................................................................................................65
Discussion ..............................................................................................................67
Conclusions ...........................................................................................................73

5 IMPROVED GLOBAL CONVERGENCE USING MULTIPLE INDEPENDENT OPTIMIZATIONS

Overview ..............................................................................................................74
Introduction ...........................................................................................................74
Methodology ..........................................................................................................77
  Analytical Test Set .........................................................................................77
  Multiple-run Methodology .............................................................................78
    Exploratory run and budgeting scheme .......................................................81
    Bayesian convergence probability estimation ..........................................84
Numerical Results .....................................................................................................85
  Multi-run Approach for Predetermined Number of Optimizations ..........85
  Multi-run Efficiency .......................................................................................87
  Bayesian Convergence Probability Estimation ............................................89
  Monte Carlo Convergence Probability Estimation ....................................92
Conclusions ..........................................................................................................92

6 PARALLELISM BY DECOMPOSITION METHODOLOGIES ................................94
Overview .............................................................................................................................. 94
Introduction ..................................................................................................................... 94
Quasiseparable Decomposition Theory ........................................................................ 96
Stepped Hollow Cantilever Beam Example .................................................................. 98
Stepped hollow beam optimization ............................................................................ 102
Quasiseparable Optimization Approach .................................................................. 104
Results ............................................................................................................................... 106
All-at-once Approach ................................................................................................. 106
Hybrid all-at-once Approach ....................................................................................... 107
Quasiseparable Approach ............................................................................................ 108
Approximation of Constraint Margins ........................................................................ 109
Discussion ...................................................................................................................... 112
Conclusions .................................................................................................................. 115

7 CONCLUSIONS ......................................................................................................... 116
Parallellism by Exploitation of Optimization Algorithm Structure ................................ 116
Parallelism through Multiple Independent Optimizations .......................................... 116
Parallellism through Concurrent Optimization of Decomposed Problems ................ 117
Future Directions .......................................................................................................... 117
Summary ....................................................................................................................... 118

APPENDIX

A ANALYTICAL TEST PROBLEM SET ..................................................................... 119
Griewank ....................................................................................................................... 119
Hartman 6 ..................................................................................................................... 119
Shekel 10 ..................................................................................................................... 120

B MONTE CARLO VERIFICATION OF GLOBAL CONVERGENCE PROBABILITY .... 122
LIST OF REFERENCES ................................................................................................. 126
BIOGRAPHICAL SKETCH ............................................................................................ 138
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Standard PSO algorithm parameters used in the study</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>Fraction of successful optimizer runs for the analytical test problems</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>Final cost function values and associated marker distance and joint parameter root-mean-square (RMS) errors after 10,000 function evaluations performed by multiple unscaled and scaled PSO, GA, SQP, and BFGS runs</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>Parallel PSO results for the biomechanical system identification problem using synthetic marker trajectories without and with numerical noise</td>
<td>66</td>
</tr>
<tr>
<td>5</td>
<td>Parallel PSO results for the biomechanical system identification problem using synthetic marker trajectories without and with numerical noise</td>
<td>67</td>
</tr>
<tr>
<td>6</td>
<td>Particle swarm algorithm parameters</td>
<td>77</td>
</tr>
<tr>
<td>7</td>
<td>Problem convergence tolerances</td>
<td>78</td>
</tr>
<tr>
<td>8</td>
<td>Theoretical convergence probability results for Hartman problem</td>
<td>87</td>
</tr>
<tr>
<td>9</td>
<td>Minimum, maximum and median fitness evaluations when applying ratio of change stopping criteria on pool of 1,000 optimizations for Griewank, Hartman and Shekel problems</td>
<td>89</td>
</tr>
<tr>
<td>10</td>
<td>Beam material properties and end load configuration</td>
<td>101</td>
</tr>
<tr>
<td>11</td>
<td>Stepped hollow beam global optimum</td>
<td>104</td>
</tr>
<tr>
<td>12</td>
<td>All-at-once approach median solution</td>
<td>107</td>
</tr>
<tr>
<td>13</td>
<td>Hybrid, all-at-once median solution</td>
<td>108</td>
</tr>
<tr>
<td>14</td>
<td>Quasiseparable optimization result</td>
<td>110</td>
</tr>
<tr>
<td>15</td>
<td>Surrogate lower level approximation optimization results</td>
<td>112</td>
</tr>
<tr>
<td>16</td>
<td>Hartman problem constants</td>
<td>120</td>
</tr>
<tr>
<td>17</td>
<td>Shekel problem constants</td>
<td>121</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Collaborative optimization flow diagram</td>
</tr>
<tr>
<td>2</td>
<td>Collaborative optimization subspace constraint satisfaction procedure (taken from [6])</td>
</tr>
<tr>
<td>3</td>
<td>Concurrent subspace optimization methodology flow diagram</td>
</tr>
<tr>
<td>4</td>
<td>Example hierarchical problem structure</td>
</tr>
<tr>
<td>5</td>
<td>Sub-problem information flow</td>
</tr>
<tr>
<td>6</td>
<td>Joint locations and orientations in the parametric ankle kinematic model</td>
</tr>
<tr>
<td>7</td>
<td>Comparison of convergence history results for the analytical test problems</td>
</tr>
<tr>
<td>8</td>
<td>Final cost function values for ten unscaled (dark bars) and scaled (gray bars) parallel PSO, GA, SQP, and BFGS runs for the biomechanical test problem</td>
</tr>
<tr>
<td>9</td>
<td>Convergence history for unscaled (dark lines) and scaled (gray lines) parallel PSO, GA, SQP, and BFGS runs for the biomechanical test problem</td>
</tr>
<tr>
<td>10</td>
<td>Sensitivity of gradient calculations to selected finite difference step size for one design variable</td>
</tr>
<tr>
<td>11</td>
<td>Serial implementation of PSO algorithm</td>
</tr>
<tr>
<td>12</td>
<td>Parallel implementation of the PSO algorithm</td>
</tr>
<tr>
<td>13</td>
<td>Surface plots of the (a) Griewank and (b) Corana analytical test problems showing the presence of multiple local minima</td>
</tr>
<tr>
<td>14</td>
<td>Average fitness convergence histories for the (a) Griewank and (b) Corana analytical test problems for swarm sizes of 16, 32, 64, and 128 particles and 10000 swarm iterations</td>
</tr>
<tr>
<td>15</td>
<td>Fitness convergence and parameter error plots for the biomechanical system identification problem using synthetic data with noise</td>
</tr>
<tr>
<td>16</td>
<td>(a) Speedup and (b) parallel efficiency for the analytical and biomechanical optimization problems</td>
</tr>
</tbody>
</table>
Multiple local minima for Griewank analytical problem surface plot in two dimensions.................................................................75
Cumulative convergence probability $P_c$ as a function of the number of optimization runs with assumed equal $P_i$ values..................................................................................81
Fitness history and convergence probability $P_c$ plots for Griewank, Hartman and Shekel problems ..................................................................................................................82
Typical Shekel fitness history plots of 20 optimizations (sampled out of 1000).....83
Shekel convergence probability for an individual optimization as a function of fitness evaluations and population size.................................................................85
Theoretical cumulative convergence probability $P_c$ as a function of the number of optimization runs with constant $P_i$ for the Hartman problem .................................................86
Theoretical convergence probability $P_c$ with sets of multiple runs for the Griewank problem ..................................................................................................................88
Theoretical convergence probability $P_c$ using information from exploratory optimizations which are stopped using a rate of change stopping condition for the Griewank, Hartman and Shekel problems.................................................................90
Bayesian $P_c$ estimation comparison to using extrapolated and randomly sampled optimizations out of pool of 1000 runs for Griewank problem.................................91
Bayesian $P_c$ estimation comparison to using extrapolated and randomly sampled optimizations out of pool of 1000 runs for Hartman problem. .................................91
Bayesian $P_c$ estimation comparison to using extrapolated and randomly sampled optimizations out of pool of 1000 runs for Shekel problem. .................................92
Stepped hollow cantilever beam .................................................................................99
Dimensional parameters of each cross section.........................................................99
Projected displacement in direction $\theta$ ....................................................................100
Tip deflection contour plot as a function of beam section 5 with height $h$, and width $w$ with yield stress and aspect ratio constraints indicated by dashed and dash dotted lines respectively.................................................................103
Quasiseparable optimization flow chart.......................................................................105
Results for 1000 all-at-once optimizations................................................................106
Hybrid PSO-fmincon strategy for 100 optimizations ..................................................108
| 35 | Repeated optimizations of section 1 subproblem using *fmincon* function. ..........110 |
| 36 | Summed budget value and constraint margins for individual sections. ..............111 |
| 37 | Global and local optimum in section 1 sub-optimization. Scale is 0.1:1 ..........111 |
| 38 | Decomposed cross section solution. Scale is 0.1:1 .......................................112 |
| 39 | Target tip deflection value histories as a function of upper-level fitness evaluations .................................................................113 |
| 40 | Constraint margin value histories as a function of upper-level function evaluations .................................................................114 |
| 41 | Predicted and Monte Carlo sampled convergence probability $P_c$ for 5 independent optimization runs for the Griewank problem ........................................122 |
| 42 | Predicted and Monte Carlo sampled convergence probability $P_c$ for 12 independent optimization runs for the Griewank problem ........................................123 |
| 43 | Monte Carlo sampled convergence probability $P_c$ with sets of multiple runs for the Griewank problem .................................................................123 |
| 44 | Monte Carlo sampled convergence probability $P_c$ using information from exploratory optimizations stopped using a rate of change stopping condition for the Griewank, Hartman and Shekel problems ........................................124 |
| 45 | Bayesian $P_c$ comparison for Griewank, Hartman and Shekel problem ..............125 |
Abstract of Dissertation Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

APPLICATIONS OF PARALLEL GLOBAL OPTIMIZATION TO MECHANICS PROBLEMS

By

Jaco Francois Schutte

December 2005

Chair: Raphael T. Haftka
Cochair: Benjamin J. Fregly
Major Department: Mechanical and Aerospace Engineering

Global optimization of complex engineering problems, with a high number of variables and local minima, requires sophisticated algorithms with global search capabilities and high computational efficiency. With the growing availability of parallel processing, it makes sense to address these requirements by increasing the parallelism in optimization strategies. This study proposes three methods of concurrent processing. The first method entails exploiting the structure of population-based global algorithms such as the stochastic Particle Swarm Optimization (PSO) algorithm and the Genetic Algorithm (GA). As a demonstration of how such an algorithm may be adapted for concurrent processing we modify and apply the PSO to several mechanical optimization problems on a parallel processing machine. Desirable PSO algorithm features such as insensitivity to design variable scaling and modest sensitivity to algorithm parameters are demonstrated. A second approach to parallelism and improving algorithm efficiency is by utilizing multiple optimizations. With this method a budget of fitness evaluations is distributed
among several independent sub-optimizations in place of a single extended optimization. Under certain conditions this strategy obtains a higher combined probability of converging to the global optimum than a single optimization which utilizes the full budget of fitness evaluations. The third and final method of parallelism addressed in this study is the use of quasiseparable decomposition, which is applied to decompose loosely coupled problems. This yields several sub-problems of lesser dimensionality which may be concurrently optimized with reduced effort.
CHAPTER 1
INTRODUCTION

Statement of Problem

Modern large scale problems often require high-fidelity analyses for every fitness evaluation. In addition to this, these optimization problems are of a global nature in the sense that many local minima exist. These two factors combine to form exceptionally demanding optimization problems which require many hours of computation on high-end single-processor computers. In order to efficiently solve such challenging problems parallelism may be employed for improved optimizer throughput on computational clusters or multi-core processors.

Purpose of Research

The research presented in this manuscript is targeted on the investigation of methods of implementing parallelism in global optimization. These methods are (i) parallel processing through the optimization algorithm, (ii) multiple independent concurrent optimizations, and (iii) parallel processing by decomposition. Related methods in the literature are reported and will be compared to the approaches formulated in this study.

Significance of Research

Parallel processing is becoming a rapidly growing resource in the engineering community. Large “processor farms” or Beowulf clusters are becoming increasingly common at research and commercial engineering facilities. In addition to this, processor manufacturers are encountering physical limitations such as heat dissipation and
constraints on processor dimensions at current clock frequencies because of the upper limit on signal speeds. These place an upper limit on the clock frequencies that can be attained and have forced manufacturers to look at other alternatives to improve processing capability. Both Intel and AMD are currently developing methods of putting multiple processors on a single die and will be releasing multiprocessor cores in the consumer market in the near future. This multi-core technology will enable even users of desktop computers to utilize concurrent processing and make it an increasingly cheap commodity in the future. The engineering community is facing more complex and computationally demanding problems as the fidelity of simulation software is improved every day. New methods that can take advantage of the increasing availability of parallel processing will give the engineer powerful tools to solve previously intractable problems. In this manuscript the specific problem of the optimization of large-scale global engineering problems is addressed by utilizing three different avenues of parallelism. Any one of these methods and even combinations of them may utilize concurrent processing to its advantage.

**Parallelism by Exploitation of Optimization Algorithm Structure**

Population-based global optimizers such as the Particle Swarm Optimizer (PSO) or Genetic Algorithms (GA’s) coordinate their search effort in the design space by evaluating a population of individuals in an iterative fashion. These iterations take the form of discrete time steps for the PSO and generations in the case of the GA. Both the PSO and the GA algorithm structures allow fitness calculations of individuals in the population to be evaluated independently and concurrently. This opens up the possibility of assigning a computational node or processor in a networked group of machines to each
individual in the population, and calculating the fitness of each individual concurrently for every iteration of the optimization algorithm.

**Parallelism through Multiple Independent Concurrent Optimizations**

A single optimization of a large-scale problem will have significant probability of becoming entrapped in a local minimum. This risk is alleviated by utilizing population based algorithms such as PSO and GA's. These global optimizers have the means of escaping from such a local minima if enough iterations are allowed. Alternatively, a larger population may be used, allowing for higher sampling densities of the design space, which also reduces the risk of entrapment. Both these options require significant additional computation effort, with no guarantee of improvement in global convergence probability. A more effective strategy can be followed while utilizing the same amount of resources. By running several independent but limited optimizations it will be shown that in most cases the combined probability of finding the global optimum is greatly improved. The limited optimization runs are rendered independent by applying a population based optimizer with different sets of initial population distributions in the design space.

**Parallelism through Concurrent Optimization of Decomposed Problems**

Some classes of such problems may be subdivided into several more tractable subproblems by applying decomposition strategies. This process of decomposition generally involves identifying groups of variables and constraints with minimal influence on one another. The choice of which decomposition strategy to apply depends largely on the original problem structure, and the interaction among variables. The objective is to find an efficient decomposition strategy to separate such large scale global optimization
problems into smaller sub-problems without introducing spurious local minima, and to apply an efficient optimizer to solve the resulting sub-problems.

**Roadmap**

A background on the global optimization of large scale optimization problems, appropriate optimization algorithms and techniques, and parallelism will be presented in Chapter 2. Chapter 3 presents an evaluation of the global, stochastic population based algorithm, the Particle Swarm Optimizer, through several analytical and biomechanical system identification problems. In Chapter 4 the parallelization of this population based algorithm is demonstrated and applied. Chapter 5 details the use of multiple independent concurrent optimizations for significant improvements in combined convergence probability. Chapter 6 shows how complex structural problems with a large number of variables may be decomposed into multiple independent sub-problems which can be optimized concurrently using a two level optimization scheme. In Chapter 7 some conclusions are drawn and avenues for future research are proposed.
CHAPTER 2
BACKGROUND

Population-based Global Optimization

Global optimization often requires specialized robust approaches. These include stochastic and/or population-based optimizers such as GAs and the PSO. The focus of this research is on exploring avenues of parallelism in population based optimization algorithms. We demonstrate these methods using the stochastic Particle Swarm Optimizer, which is a stochastic search algorithm suited to continuous problems. Other merits to the PSO include low sensitivity to algorithm parameters, and insensitivity to scaling of design variables. These qualities will be investigated in Chapter 3. This algorithm does not require gradients, which is an important consideration when solving problems of high dimensionality, often the case in large scale optimization. The PSO has a performance comparable to GAs, which are also candidates for any of the methods of parallelism proposed in this manuscript, and may be more suitable for discrete or mixed variable types of problems.

In the research presented in this manuscript the PSO is applied to a biomechanical problem with a large number of continuous variables. This problem has several local minima, and, when attempting to solve it with gradient based optimizers, demonstrated high sensitivity to the scaling of design variables. This made it an ideal candidate to demonstrate the desirable quantities of the algorithm. Other application problems include structural sizing problems and composite laminate angle optimization.
Parallel Processing in Optimization

There are five approaches which may be utilized to decompose a single computational task into smaller problems which may then be solved concurrently. These are geometric decomposition, iterative decomposition, recursive decomposition, speculative decomposition and functional decomposition, or a combination of these [1,2]. Among these, functional decomposition is most commonly applied and will also be the method of implementation presented in Chapter 4. The steps followed in parallelizing a sequential program consist of the following from [1].

1. Decompose the sequential program or data into smaller tasks.
2. Assign the tasks to processes. A process is an abstract entity that performs tasks.
3. Orchestrate the necessary data access, communication, and synchronization.
4. Map or assign the processes to computational nodes.

The functional decomposition method is based on the premise that applications such as an optimization algorithm may be broken into many distinct phases, each of which interacts with some or all of the others. These phases can be implemented as co-routines, each of which will execute for as long as it is able and then invoke another and remain suspended until it is again needed. Functional decomposition is the simplest way of parallelization if it can be implemented by turning its high-level description into a set of cooperating processes [2]. When using this method, balancing the throughput of the different computational stages will be highly problematic when there are dependencies between stages, for example, when data requires sequential processing by several stages. This limits the parallelism that may be achieved using functional decomposition. Any further parallelism must be achieved through using geometric, iterative or speculative decomposition within a functional unit [2].
When decomposing a task into concurrent processes some additional communication among these routines is required for coordination and the interchange of data. Among the methods of communication for parallel programming the parallel virtual machine (PVM) and the message-passing interface (MPI) are the most widely used. For the research undertaken in Chapter 4 a portable implementation of the MPI library [3,4] containing a set of parallel communication functions [5] is used.

**Decomposition in Large Scale Optimization**

The optimization community developed several formalized decomposition methods such as Collaborative Optimization (CO) [6], Concurrent SubSpace Optimization (CSSO) [7], and Quasiseparable decomposition to deal with the challenges presented by large scale engineering problems. The problems addressed using these schemes include multi-disciplinary optimization in aerospace design, or large scale structural and biomechanical problems.

Decomposition methodologies in large scale optimization are currently intensely studied because increasingly advanced higher fidelity simulation methods result in large scale problems becoming intractable. Problem decomposition allows for:

1. Simplified decomposed subsystems. In most cases the decomposed sub-problems are of reduced dimensionality, and therefore less demanding on optimization algorithms. An example of this is the number of gradient calculations required per optimization iteration, which in the case of gradient based algorithms, scales directly with problem dimensionality.

2. A broader work front to be attacked simultaneously, which results in a problem being solved in less time if the processing resources are available. Usually computational throughput is limited in a sequential fashion, i.e., the FLOPS limit for computers. However, if multiple processing units are available this limit can be circumvented by using an array of networked computers, for example, a Beowulf cluster.
Several such decomposition strategies have been proposed (see next section for a short review), all differing in the manner in which they address some or all of the following.

1. Decomposition boundaries, which may be disciplinary, or component interfaces in a large structure.
2. Constraint handling
3. Coordination among decomposed sub-problems.

**Literature Review: Problem Decomposition Strategies**

Here follows a summary of methodologies used for the decomposition and optimization of large scale global problems. This review forms the background for the study proposed in Chapter 6 of this manuscript.

**Collaborative Optimization (CO)**

**Overview.** The Collaborative Optimization (CO) strategy was first introduced by Kroo et al., [8]. Shortly after its introduction this bi-level optimization scheme was extended by Tappeta and Renaud [9,10] to three distinct formulations to address multi-objective optimization of large-scale systems. The CO paradigm is based on the concept that the interaction among several different disciplinary experts optimizing a design is minimal for local changes in each discipline in a MDO problem. This allows a large scale system to be decomposed into sub-systems along domain specific boundaries. These subsystems are optimized through local design variables specific to each subsystem, subject to the domain specific constraints. The objective of each subsystem optimization is to maintain agreement on interdisciplinary design variables. A system level optimizer enforces this interdisciplinary compatibility while minimizing the overall objective function. This is achieved by combining the system level fitness with the cumulative sum
of all discrepancies between interdisciplinary design variables. This strategy is extremely suited for parallel computation because of the minimal interaction between the different design disciplines, which results in reduced communication overhead during the course of the optimization.

**Methodology.** This decomposition strategy is described with the flow diagram in Figure 1. As mentioned previously, the CO strategy is a bi-level method, in which the system level optimization sets and adjusts a interdisciplinary design variables during the optimization. The subspace optimizer attempts both to satisfy local constraints by adjusting local parameters, and to meet the interdisciplinary design variable targets set by the system level optimizer. Departures from the target interdisciplinary design parameters are allowed, which may occur because of insufficient local degrees of freedom, but is to be minimized. The system level optimizer attempts to adjust the interdisciplinary parameters such that the objective function is minimized, while maximizing the agreement between subsystems. This process of adjusting the system level target design, and the subsystems attempting to match it whilst satisfying local constraints, is repeated until convergence.

This procedure can be graphically illustrated in Figure 2 (taken from Kroo and Manning [6]). The system level optimizer sets a design target P, and each subspace optimization attempts to satisfy local constraints while matching the target design P as closely as possible by moving in directions 1 and 2. During the next system level optimization cycle the target design P will be moved in direction 3 in order to maximize the agreement between the target design and subspace designs that satisfy local constraints.
Refinements on method. Several enhancements to this method have been proposed, among which are the integration of this architecture into a decision based design framework as proposed by Hazelrigg [11,12], the use of response surfaces [13] to
model disciplinary analyses, and genetic algorithms with scheduling for increased efficiency [14].

**Solution quality and computational efficiency.** Sobieski and Kroo [15] report very robust performance on their CO scheme, with identical solutions being found on both collaborative and single level optimizations of 45 design variable, 26 constraint problems.

Braun and Kroo [16] showed CO to be unsuited for small problems with strong coupling, but for large scale problems with weak coupling the CO methodology becomes more computationally efficient. They also found that the amount of system level iterations is dependent on the level of coupling of sub-systems, and that the required number of sub-optimizations scales in proportion to the overall problem size. Similar findings are reported by Alexandrov [17]. Braun et al. [18] evaluated the performance of CO on a set of quadratic problems presented by Shankar et al. [19] to evaluate the CSSO method. Unlike CSSO, the CO method did not require an increased amount of iterations for QP problems with strong coupling, and converged successfully in all cases.

**Applications.** This decomposition architecture has been extensively demonstrated using analytical test problems [18,20] and aerospace optimization problems such as trajectory optimization [16,18], vehicle design [13,20-22], and satellite constellation configurations [23].

**Concurrent SubSpace Optimization (CSSO)**

**Overview.** This method was proposed by Sobieszczanski-Sobieski [24], and, like CO, divides the MDO problem along disciplinary boundaries. The main difference however is the manner in which the CSSO framework coordinates the subsystem optimizations. A bi-level optimization scheme is used in which the upper optimization
problem consists of a linear [7] or second order [25] system approximation created with the use of Global Sensitivity Equations (GSE’s). This system approximation reflects changes in constraints and the objective function as a function of design variables. Because of the nonlinearities, this approximation is only accurate in the immediate neighborhood of the current design state, and needs to be updated after every upper-level optimization iteration. After establishing a system level approximation the subsystems are independently optimized using only design variables local to the subspace. The system level approximation is then updated by a sensitivity analysis to reflect changes in the subspace design. The last two steps of subspace optimization and system approximation is repeated through the upper level optimizer until convergence is achieved.

**Methodology.** The basic steps taken to optimize a MDO problem with the CSSO framework are as follows:

1. Select initial set of designs for each subsystem.
2. Construct system approximation using GSE’s
3. a) Subspace optimization through local variables and objective
   b) Update system approximation by performing sensitivity analysis
4. Optimize design variables according to system approximation
5. Stop if converged; otherwise go to 3)

where step (3) contains the lower level optimization in this bi-level framework. This process is illustrated in Figure 3.

**Refinements on method.** As previously mentioned, early CSSO strategies used linear system level approximations obtained with the Global Sensitivity Equations (GSE). Coordination of subspace or disciplinary optimizations is achieved through system level sensitivity information.
This imposes limits on the allowable deviation from the current design, and requires a new approximation to be constructed at every system iteration to maintain reasonable accuracy. Recent research focused on alternate methods for acquiring system level approximations for the coordination effort. Several authors [25-28] modified the system approximation to utilize a second order response surface approximation. This is combined with a database of previous fitness evaluation points, which can be used to create and update the response surface. This response surface then serves to couple the subsystem optimizations and coordinate system level design.

**Solution quality and computational efficiency.** Shankar et al. [19] investigated the robustness of the CSSO on a set of analytical problems. Several quadratic programming problems with weak and strong coupling between subsystems were
evaluated with a modification of Sobieszczanski-Sobieski’s nonhierarchical subspace optimization scheme [7]. Results indicated reasonable performance for problems with weak coupling between subsystems. For large problems with strong interactions between subsystems, this decomposition scheme proved unreliable in terms of finding global sensitivities, leading to poor solutions.

Tappeta et al., using the iSIGHT software. [29-31], analyzed two analytical and two structural problems, a welding design and a stepped beam weight minimization. In this work it is reported that the Karush-Kuhn-Tucker conditions were met in some of the cases, and that most problems converged closely to the original problem solution.

Lin and Renaud compared the commercial software package LANCELOT [32] which compares the Broydon-Fletcher-Goldfarb-Shanno (BFGS) method to the CSSO strategy, the latter incorporating response surfaces. In this study the authors show similar computational efficiencies for small uncoupled analytical problems. For large scale MDO problems however the CSSO method consistently outperformed the LANCELOT optimizer in this area.

Sellar et al. [26] compared a CSSO with neural network based response surface enhancements with a full (all at once) system optimization. The CSSO-NN algorithm showed a distinct advantage in computational efficiency over the all-at-once approach, while maintaining a high level of robustness.

**Applications.** This decomposition methodology has been applied to large scale aerospace problems like high temperature and pressure aircraft engine components [29,33], aircraft brake component optimization [34], and aerospace vehicle design [26,35].
Analytical Target Cascading (ATC)

Overview. Analytical Target Cascading was introduced by Michelena et al. [36] in 1999, and developed further by Kim [37] as a product development tool. This method is typically used to solve object based decomposed system optimization problems. Tosserams et al. [38] introduced a Lagrangian relaxation strategy which in some cases improves the computational efficiency of this method by several orders of magnitude.

Methodology. ATC is a strategy which coordinates hierarchically decomposed systems or elements of a problem (see Figure 4) by the introduction of target and response coupling variables. Targets are set by parent elements which are met by responses from the children elements in the hierarchy (see Figure 5 obtained from [38]). At each element an optimization problem is formulated to find local variables, parent responses and child targets which minimize a penalized discrepancy function, while meeting local constraints. The responses are rebalanced up to higher levels by iteratively changing targets in a nested loop in order to obtain consistency. Several coordination strategies are available to determine the sequence of solving the sub-problems and order of exchange in targets and responses [39]. Proof of convergence is also presented for some of these classes of approaches in [39].

Refinements on method. Tosserams et al. [38] introduced the use of augmented Lagrangian relaxation in order to reduce the computational cost associated with obtaining very accurate agreement between sub-problems, and the coordination effort at the inner loop of the method. Allison et al. exploited the complimentary nature of ATC and CO to obtain an optimization formulation called nested ATC-MDO [40]. Kokkolaras et al. extended the formulation of ATC to include the design of product families [41].
Solution quality and computational efficiency. Michelena et al. [39] prove, under certain convexity assumptions that the ATC process will yield the optimal solution of the original design target problem. The original ATC formulation had the twofold problem of requiring large penalty weights to accurate solutions, and the excessive repeat in the inner loop which solves sub-problems before the outer loop can proceed. Both these problems are addressed by using augmented Lagrangian relaxation [38] by Tosserams et al., which reported a decrease in computational effort on the order of between orders 10 and 10000.
Applications. The ATC strategy is applied the design of structural members and a electric water pump in [40], and automotive design [42,43]. The performance of the Augmented Lagrangian Relaxation ATC enhancement was tested using several geometric programming problems [38].

Quasiseparable Decomposition and Optimization

The quasiseparable decomposition and optimization strategy will be the focus of the research proposed in Chapter 6. This methodology addresses a class of problems common in the field of engineering and can be applied to a wide class of structural, biomechanical and other disciplinary problems. The strategy, which will be explained in detail in Chapter 6, is based on a two-level optimization approach which allows for the global search effort to be concentrated at the lower level sub-problem optimizations. The system to be optimized is decomposed in to several lower level subsystem optimizations which are coordinated by an upper level optimization. A Sequential Quadratic Programming (SQP) based optimizer is applied in this optimization infrastructure to solve a example structural sizing problem. This example problem entails the maximization of the tip displacement of a hollow stepped cantilever beam with 5 sections. The quasiseparable decomposition methodology is applied to decompose the structure into several sub-problems of reduced dimensionality. The parallelism with this strategy is achieved by optimizing the independent sub-problems (sections of the stepped beam) concurrently, allowing for the utilization of parallel processing resources.
CHAPTER 3
GLOBAL OPTIMIZATION THROUGH THE PARTICLE SWARM ALGORITHM

Overview

This chapter introduces the population based algorithm which will be the target for investigating parallelism throughout the manuscript. This stochastic algorithm mimics swarming or flocking behavior found in animal groups such as bees, fish and birds. The swarm of particles is basically several parallel individual searches that are influenced by individual and swarm memory of regions in the design space with high fitness. These positions of the regions are constantly updated and reported to the individuals in the swarm through a simple communication model. This model allows for the algorithm to be easily decomposed into concurrent processes, each representing an individual particle in the swarm. This approach to allow parallelism will be detailed in Chapter 4.

For the purpose of illustrating the performance of the PSO, it is compared to several other algorithms commonly used when solving problems in biomechanics. This comparison is made through optimization of several analytical problems, and a biomechanical system identification problem. The focus of the research presented in this chapter is to demonstrate that the PSO has good properties such as insensitivity to the scaling of design variables and very few algorithm parameters to fine tune. These make the PSO a valuable addition in the arsenal of optimization methods in biomechanical optimization, in which it has never been applied before.

The work presented in this Chapter was in collaboration with Jeff Reinbold, who supplied the biomechanical test problem [58] and Byung Il Koh, who developed the
parallel SQP and BFGS algorithms [77] used to establish a computational efficiency comparison. The research in this Chapter was also published in [58,76,129]. Thanks goes to Soest and Casius for their willingness to share their numerical results published in [48].

**Introduction**

Optimization methods are used extensively in biomechanics research to predict movement-related quantities that cannot be measured experimentally. Forward dynamic, inverse dynamic, and inverse static optimizations have been used to predict muscle, ligament, and joint contact forces during experimental or predicted movements (e.g., see references [44-55]). System identification optimizations have been employed to tune a variety of musculoskeletal model parameters to experimental movement data (e.g., see references [56-60]). Image matching optimizations have been performed to align implant and bone models to *in vivo* fluoroscopic images collected during loaded functional activities (e.g., see references [61-63]).

Since biomechanical optimization problems are typically nonlinear in the design variables, gradient-based nonlinear programming has been the most widely used optimization method. The increasing size and complexity of biomechanical models has also led to parallelization of gradient-based algorithms, since gradient calculations can be easily distributed to multiple processors [44-46]. However, gradient-based optimizers can suffer from several important limitations. They are local rather than global by nature and so can be sensitive to the initial guess. Experimental or numerical noise can exacerbate this problem by introducing multiple local minima into the problem. For some problems, multiple local minima may exist due to the nature of the problem itself. In most situations, the necessary gradient values cannot be obtained analytically, and finite
difference gradient calculations can be sensitive to the selected finite difference step size. Furthermore, the use of design variables with different length scales or units can produce poorly scaled problems that converge slowly or not at all [64,65], necessitating design variable scaling to improve performance.

Motivated by these limitations and improvements in computer speed, recent studies have begun investigating the use of non-gradient global optimizers for biomechanical applications. Neptune [47] compared the performance of a simulated annealing (SA) algorithm with that of downhill simplex (DS) and sequential quadratic programming (SQP) algorithms on a forward dynamic optimization of bicycle pedaling utilizing 27 design variables. Simulated annealing found a better optimum than the other two methods and in a reasonable amount of CPU time. More recently, Soest and Casius [48] evaluated a parallel implementation of a genetic algorithm (GA) using a suite of analytical tests problems with up to 32 design variables and forward dynamic optimizations of jumping and isokinetic cycling with up to 34 design variables. The genetic algorithm generally outperformed all other algorithms tested, including SA, on both the analytical test suite and the movement optimizations.

This study evaluates a recent addition to the arsenal of global optimization methods – particle swarm optimization (PSO) – for use on biomechanical problems. A recently-developed variant of the PSO algorithm is used for the investigation. The algorithm’s global search capabilities are evaluated using a previously published suite of difficult analytical test problems with multiple local minima [48], while its insensitivity to design variable scaling is proven mathematically and verified using a biomechanical test problem. For both categories of problems, PSO robustness, performance, and scale-
independence are compared to that of three off-the-shelf optimization algorithms - a genetic algorithm (GA), a sequential quadratic programming algorithm (SQP), and the BFGS quasi-Newton algorithm. In addition, previously published results [48] for the analytical test problems permit comparison with a more complex GA algorithm (GA*), a simulated annealing algorithm (SA), a different SQP algorithm (SQP*), and a downhill simplex (DS) algorithm.

Theory

Particle Swarm Algorithm.

Particle swarm optimization is a stochastic global optimization approach introduced by Kennedy and Eberhart [66]. The method's strength lies in its simplicity, being easy to code and requiring few algorithm parameters to define convergence behavior. The following is a brief introduction to the operation of the particle swarm algorithm based on a recent implementation by Groenwold and Fourie [67] incorporating dynamic inertia and velocity reduction.

Consider a swarm of \( p \) particles, where each particle's position \( x^i_k \) represents a possible solution point in the problem design space \( D \). For each particle \( i \), Kennedy and Eberhart [66] proposed that the position \( x^i_{k+1} \) be updated in the following manner:

\[
x_{k+1} = x_k + v_{k+1}
\]  

(3.1)

with a pseudo-velocity \( v^i_{k+1} \) calculated as follows:

\[
v^i_{k+1} = w^i_k v^i_k + c_1 r_1 (p^i_k - x^i_k) + c_2 r_2 (g_k - x^i_k)
\]  

(3.2)

Here, subscript \( k \) indicates a (unit) pseudo-time increment. The point \( p^i_k \) is the best-found cost location by particle \( i \) up to timestep \( k \), which represents the cognitive contribution to the search vector \( v^i_{k+1} \). Each component of \( v^i_{k+1} \) is constrained to be less
than or equal to a maximum value defined in $v_{k+1}^{\text{max}}$. The point $g_k$ is the global best-found position among all particles in the swarm up to time $k$ and forms the social contribution to the velocity vector. Cost function values associated with $p_k^i$ and $g_k$ are denoted by $f_{\text{best}}^i$ and $f_{\text{best}}^g$ respectively. Random numbers $r_1$ and $r_2$ are uniformly distributed in the interval [0,1]. Shi and Eberhart [68] proposed that the cognitive and social scaling parameters $c_1$ and $c_2$ be selected such that $c_1 = c_2 = 2$ to allow the product $c_1r_1$ or $c_2r_2$ to have a mean of 1. The result of using these proposed values is that the particles overshoot the attraction points $p_k^i$ and $g_k$ half the time, thereby maintaining separation in the group and allowing a greater area to be searched than if the particles did not overshoot. The variable $w_k$, set to 1 at initialization, is a modification to the original PSO algorithm [66]. By reducing its value dynamically based on the cost function improvement rate, the search area is gradually reduced [69]. This dynamic reduction behavior is defined by $w_d$, the amount by which the inertia $w_k$ is reduced, $v_d$, the amount by which the maximum velocity $v_{k+1}^{\text{max}}$ is reduced, and $d$, the number of iterations with no improvement in $g_k$ before these reduction take place [67] (see algorithm flow description below).

Initialization of the algorithm involves several important steps. Particles are randomly distributed throughout the design space, and particle velocities $v_0^i$ are initialized to random values within the limits $0 \leq v_0^i \leq v_0^{\text{max}}$. The particle velocity upper limit $v_0^{\text{max}}$ is calculated as a fraction of the distance between the upper and lower bound on variables in the design space $v_0^{\text{max}} = \kappa(x_{UB} - x_{LB})$ with $\kappa = 0.5$ as suggested in [69]. Iteration counters $k$ and $t$ are set to 0. Iteration counter $k$ is used to monitor the total
number of swarm iterations, while iteration counter $t$ is used to monitor the number of
swarm iterations since the last improvement in $g_k$. Thus, $t$ is periodically reset to zero
during the optimization while $k$ is not.

The algorithm flow can be represented as follows:

1. Initialize
   a. Set constants $\kappa, c_1, c_2, k_{\text{max}}, v_0^{\text{max}}, w_0, v_d, w_d$, and $d$
   b. Set counters $k = 0, t = 0$. Set random number seed.
   c. Randomly initialize particle positions $x_0^i \in \mathcal{R}$ for $i = 1, \ldots, p$
   d. Randomly initialize particle velocities $0 \leq v_0^i \leq v_0^{\text{max}}$ for $i = 1, \ldots, p$
   e. Evaluate cost function values $f_0^i$ using design space coordinates $x_0^i$ for $i = 1, \ldots, p$
   f. Set $f_{\text{best}}^i = f_0^i$ and $p_0^i = x_0^i$ for $i = 1, \ldots, p$
   g. Set $f_{\text{best}}^g$ to best $f_{\text{best}}^i$ and $g_0$ to corresponding $x_0^i$

2. Optimize
   h. Update particle velocity vectors $v_{k+1}^i$ using Eq. (3.2)
   i. If $v_{k+1}^i > v_{k+1}^{\text{max}}$ for any component, then set that component to its maximum
      allowable value
   j. Update particle position vectors $x_{k+1}^i$ using Eq. (3.1)
   k. Evaluate cost function values $f_{k+1}^i$ using design space coordinates $x_{k+1}^i$ for $i = 1, \ldots, p$
   l. If $f_{k+1}^i \leq f_{\text{best}}^i$, then $f_{\text{best}}^i = f_{k+1}^i$, $p_{k+1}^i = x_{k+1}^i$ for $i = 1, \ldots, p$
m. If $f_{k+1}^i \leq f_{\text{best}}^g$, then $f_{\text{best}}^g = f_{k+1}^i$, $g_{k+1}^i = x_{k+1}^i$ for $i = 1, \ldots, p$

n. If $f_{\text{best}}^g$ was improved in (e), then reset $t = 0$. Else increment $t$

o. If maximum number of function evaluations is exceeded, then go to 3

p. If $t = d$, then multiply $w_{k+1}$ by $(1 - w_d)$ and $v_{k+1}^\text{max}$ by $(1 - v_d)$

q. Increment $k$

r. Go to 2(a).

3. Report results

4. Terminate

This algorithm was coded in the C programming language by the author [70] and used for all PSO analyses performed in the study. A standard population size of 20 particles was used for all runs, and other algorithm parameters were also selected based on standard recommendations (Table 1) [70-72]. The C source code for our PSO algorithm is freely available at http://www.mae.ufl.edu/~fregly/downloads/pso.zip (last accessed 12/2005).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>Population size (number of particles)</td>
<td>20</td>
</tr>
<tr>
<td>$c_1$</td>
<td>Cognitive trust parameter</td>
<td>2.0</td>
</tr>
<tr>
<td>$c_2$</td>
<td>Social trust parameter</td>
<td>2.0</td>
</tr>
<tr>
<td>$w_0$</td>
<td>Initial inertia</td>
<td>1</td>
</tr>
<tr>
<td>$w_d$</td>
<td>Inertia reduction parameter</td>
<td>0.01</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>Bound on velocity fraction</td>
<td>0.5</td>
</tr>
<tr>
<td>$v_d$</td>
<td>Velocity reduction parameter</td>
<td>0.01</td>
</tr>
<tr>
<td>$d$</td>
<td>Dynamic inertia/velocity reduction delay (function evaluations)</td>
<td>200</td>
</tr>
</tbody>
</table>

Analysis of Scale Sensitivity.

One of the benefits of the PSO algorithm is its insensitivity to design variable scaling. To prove this characteristic, we will use a proof by induction to show that all
particles follow an identical path through the design space regardless of how the design variables are scaled. In actual PSO runs intended to investigate this property, use of the same random seed in scaled and unscaled cases will ensure that an identical sequence of random $r_1$ and $r_2$ values are produced by the computer throughout the course of the optimization.

Consider an optimization problem with $n$ design variables. An $n$-dimensional constant scaling vector $\zeta$ can be used to scale any or all dimensions of the problem design space:

$$\zeta = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \vdots \\ \zeta_n \end{bmatrix}$$

(3.3)

We wish to show that for any time step $k \geq 0$,

$$v_k' = \zeta v_k$$

(3.4)

$$x_k' = \zeta x_k$$

(3.5)

where $x_k$ and $v_k$ (dropping superscript $i$) are the unscaled position and velocity, respectively, of an individual particle and $x_k' = \zeta x_k$ and $v_k' = \zeta v_k$ are the corresponding scaled versions.

First, we must show that our proposition is true for the base case, which involves initialization ($k = 0$) and the first time step ($k = 1$). Applying the scaling vector $\zeta$ to an individual particle position $x_0$ during initialization produces a scaled particle position $x_0'$:

$$x_0' = \zeta x_0$$

(3.6)

where the right hand side is a component-by-component product of vectors $\zeta$ and $x_0$. This implies that
in the unscaled case, the pseudo-velocity is calculated as
\[ v_0 = \kappa (x_{UB} - x_{LB}) \]  
(3.8)

In the scaled case, this becomes
\[ v'_0 = \kappa (x'_{UB} - x'_{LB}) = \kappa (\zeta x_{UB} - \zeta x_{LB}) = \zeta v_0 \]  
(3.9)

From Eqs. (3.1) and (3.2) and these initial conditions, the particle pseudo-velocity and position for the first time step can be written as
\[ v_1 = w_0 v_0 + c_1 r_1 (p_0 - x_0) + c_2 r_2 (g_0 - x_0) \]  
(3.10)
\[ x_1 = x_0 + v_1 \]  
(3.11)
in the unscaled case and
\[ v'_1 = w_0 v'_0 + c_1 r_1 (p'_0 - x'_0) + c_2 r_2 (g'_0 - x'_0) = \zeta [w_0 v_0 + c_1 r_1 (\zeta p_0 - \zeta x_0) + c_2 r_2 (\zeta g_0 - \zeta x_0)] = \zeta v_1 \]  
(3.12)
\[ x'_1 = x'_0 + v'_1 = \zeta x_0 + \zeta v_1 = \zeta [x_0 + v_1] \]  
(3.13)
in the scaled case. Thus, our proposition is true for the base case.

Next, we must show that our proposition is true for the inductive step. If we assume our proposition holds for any time step \( k = j \), we must prove that it also holds for time step \( k = j + 1 \). We begin by replacing subscript \( k \) with subscript \( j \) in Eqs. (3.4) and (3.5). If we then replace subscript 0 with subscript \( j \) and subscript 1 with subscript \( j + 1 \) in Eqs. (3.12) and (3.13), we arrive at Eqs. (3.4) and (3.5) where subscript \( k \) is replaced by subscript \( j + 1 \). Thus, our proposition is true for any time step \( j + 1 \).
Consequently, since the base case is true and the inductive step is true, Eqs. (3.4) and (3.5) are true for all \( k \geq 0 \). From Eqs. (3.4) and (3.5), we can conclude that any linear scaling of the design variables (or subset thereof) will have no effect on the final or any intermediate result of the optimization, since all velocities and positions are scaled accordingly. This fact leads to identical step intervals being taken in the design space for scaled and unscaled version of the same problem, assuming infinite precision in all calculations.

In contrast, gradient-based optimization methods are often sensitive to design variable scaling due to algorithmic issues and numerical approximations. First derivative methods are sensitive because of algorithmic issues, as illustrated by a simple example. Consider the following minimization problem with two design variables \((x,y)\) where the cost function is

\[
x^2 + \frac{y^2}{100}
\]  

with initial guess \((1,1)\). A scaled version of the same problem can be created by letting \(\tilde{x} = x, \tilde{y} = y/10\) so that the cost function becomes

\[
\tilde{x}^2 + \tilde{y}^2
\]  

with initial guess \((1,10)\). Taking first derivatives of each cost function with respect to the corresponding design variables and evaluating at the initial guesses, the search direction for the unscaled problem is along a line rotated 5.7º from the positive \(x\) axis and for the scaled problem along a line rotated 45º. To reach the optimum in a single step, the unscaled problem requires a search direction rotated 84.3º and the scaled problem 45º. Thus, the scaled problem can theoretically reach the optimum in a single step while the unscaled problem cannot due to the effect of scaling on the calculated search direction.
Second derivative methods are sensitive to design variable scaling because of numerical issues related to approximation of the Hessian (second derivative) matrix. According to Gill et al. [64], Newton methods utilizing an exact Hessian matrix will be insensitive to design variable scaling as long as the Hessian matrix remains positive definite. However, in practice, exact Hessian calculations are almost never available, necessitating numerical approximations via finite differencing. Errors in these approximations result in different search directions for scaled versus unscaled versions of the same problem. Even a small amount of design variable scaling can significantly affect the Hessian matrix so that design variable changes of similar magnitude will not produce comparable magnitude cost function changes [64]. Common gradient-based algorithms that employ an approximate Hessian include Newton and quasi-Newton nonlinear programming methods such as BFGS, SQP methods, and nonlinear least-squares methods such as Levenberg-Marquardt [64]. A detailed discussion of the influence of design variable scaling on optimization algorithm performance can be found in Gill et al. [64].

**Methodology**

**Optimization Algorithms**

In addition to our PSO algorithm, three off-the-shelf optimization algorithms were applied to all test problems (analytical and biomechanical - see below) for comparison purposes. One was a global GA algorithm developed by Deb [73-75]. This basic GA implementation utilizes one mutation operator and one crossover operator along with real encoding to handle continuous variables. The other two algorithms were commercial implementations of gradient-based SQP and BFGS algorithms (VisualDOC, Vanderplaats R & D, Colorado Springs, CO).
All four algorithms (PSO, GA, SQP, and BFGS) were parallelized to accommodate the computational demands of the biomechanical test problem. For the PSO algorithm, parallelization was performed by distributing individual particle function evaluations to different processors as detailed by the author in [76]. For the GA algorithm, individual chromosome function evaluations were parallelized as described in [48]. Finally, for the SQP and BFGS algorithms, finite difference gradient calculations were performed on different processors as outlined by Koh et al. in [77]. A master-slave paradigm using the Message Passing Interface (MPI) [3,4] was employed for all parallel implementations. Parallel optimizations for the biomechanical test problem were run on a cluster of Linux-based PCs in the University of Florida High-performance Computing and Simulation Research Laboratory (1.33 GHz Athlon CPUs with 256MB memory on a 100Mbps switched Fast Ethernet network).

While the PSO algorithm used standard algorithm parameters for all optimization runs, minor algorithm tuning was performed on the GA, SQP, and BFGS algorithms for the biomechanical test problem. The goal was to give these algorithms the best possible chance for success against the PSO algorithm. For the GA algorithm, preliminary optimizations were performed using population sizes ranging from 40 to 100. It was found that for the specified maximum number of function evaluations, a population size of 60 produced the best results. Consequently, this population size was used for all subsequent optimization runs (analytical and biomechanical). For the SQP and BFGS algorithms, automatic tuning of the finite difference step size (FDSS) was performed separately for each design variable. At the start of each gradient-based run, forward and central difference gradients were calculated for each design variable beginning with a
relative FDSS of $10^{-1}$. The step size was then incrementally decreased by factors of ten until the absolute difference between forward and central gradient results was a minimum. This approach was taken since the amount of noise in the biomechanical test problem prevented a single stable gradient value from being calculated over a wide range of FDSS values (see Discussion). The forward difference step size automatically selected for each design variable was used for the remainder of the run.

**Analytical Test Problems**

The global search capabilities of our PSO implementation were evaluated using a suite of difficult analytical test problems previously published by Soest and Casius [48]. In that study, each problem in the suite was evaluated using four different optimizers: SA, GA*, SQP*, and DS, where a star indicates a different version of an algorithm used in our study. One thousand optimization runs were performed with each optimizer starting from random initial guesses and using standard optimization algorithm parameters. Each run was terminated based on a pre-defined number of function evaluations for the particular problem being solved. We followed an identical procedure with our four algorithms to permit comparison between our results and those published by Soest and Casius in [48]. Since two of the algorithms used in our study (GA and SQP) were of the same general category as algorithms used by Soest and Casius in [48] (GA* and SQP*), comparisons could be made between different implementations of the same general algorithm. Failed PSO and GA runs were allowed to use up the full number of function evaluations, whereas failed SQP and BFGS runs were re-started from new random initial guesses until the full number of function evaluations was completed. Only 100 rather than 1000 runs were performed with the SQP and BFGS algorithms due to a database size problem in the VisualDOC software.
A detailed description of the six analytical test problems can be found in Soest and Casius [48]. Since the design variables for each problem possessed the same absolute upper and lower bound and appeared in the cost function in a similar form, design variable scaling was not an issue in these problems. The six analytical test problems are described briefly below.

\( H_1 \): This simple 2-dimensional function [48] has several local maxima and a global maximum of 2 at the coordinates (8.6998, 6.7665).

\[
H_1(x_1, x_2) = \frac{\sin^2 \left( x_1 - \frac{x_2}{8} \right) + \sin^2 \left( x_2 - \frac{x_1}{8} \right)}{d + 1}
\]

where

\[
d = \sqrt{(x_1 - 8.6998)^2 + (x_2 - 6.7665)^2}
\]

Ten thousand function evaluations were used for this problem.

\( H_2 \): This inverted version of the F6 function used by Schaffer et al. [78] has 2 dimensions with several local maxima around the global maximum of 1.0 at (0,0).

\[
H_2(x_1, x_2) = 0.5 - \frac{\sin^2 \left( \sqrt{x_1^2 + x_2^2} \right) - 0.5}{\left( 1 + 0.001 \left( \sqrt{x_1^2 + x_2^2} \right) \right)^2}
\]

where

\[
x_1, x_2 \in [-100,100]
\]

This problem was solved using 20,000 function evaluations per optimization run.

\( H_3 \): This test function from Corana et al. [79] was used with dimensionality \( n = 4, 8, 16, \) and 32. The function contains a large number of local minima (on the order of \( 10^{4n} \)) with a global minimum of 0 at \( |x_i| < 0.05 \).
\[ H_3(x_1, \ldots, x_n) = \sum_{i=1}^{n} \begin{cases} (t \cdot \text{sgn}(z_i) + z_i)^2 \cdot c \cdot d_i & \text{if } |x_i - z_i| < t \\ d_i \cdot x_i^2 & \text{otherwise} \end{cases} \]  

where

\[ z_i = \left\lfloor \frac{x_i}{s} \right\rfloor + 0.49999 \cdot \text{sgn}(x_i) \cdot s, \quad c = 0.15, \]

\[ s = 0.2, \quad t = 0.05, \quad \text{and } d_i = \begin{cases} 1 & i = 1, 5, 9, \ldots \\ 1000 & i = 2, 6, 10, \ldots \\ 10 & i = 3, 7, 11, \ldots \\ 100 & i = 4, 8, 12, \ldots \end{cases} \]

The use of the floor function in Eq. (3.18) makes the search space for this problem the most discrete of all problems tested. The number of function evaluations used for this problem was 50,000 \((n = 4)\), 100,000 \((n = 8)\), 200,000 \((n = 16)\), and 400,000 \((n = 32)\).

For all of the analytical test problems, an algorithm was considered to have succeeded if it converged to within \(10^{-3}\) of the known optimum cost function value within the specified number of function evaluations [48].

**Biomechanical Test Problem**

In addition to these analytical test problems, a biomechanical test problem was used to evaluate the scale-independent nature of the PSO algorithm. Though our PSO algorithm is theoretically insensitive to design variable scaling, numerical round-off errors and implementation details could potentially produce a scaling effect. Running the other three algorithms on scaled and unscaled versions of this test problem also permitted investigation of the extent to which other algorithms are influenced by design variable scaling.

The biomechanical test problem involved determination of an ankle joint kinematic model that best matched noisy synthetic (i.e., computer generated) movement data.
Similar to that used by van der Bogert et al. [56], the ankle was modeled as a three-dimensional linkage with two non-intersecting pin joints defined by 12 subject-specific parameters (Figure 6).

Figure 6  Joint locations and orientations in the parametric ankle kinematic model. Each $p_i (i = 1, ..., 12)$ represents a different position or orientation parameter in the model.
These parameters represent the positions and orientations of the talocrural and subtalar joint axes in the tibia, talus, and calcaneous. Position parameters were in units of centimeters and orientation parameters in units of radians, resulting in parameter values of varying magnitude. This model was part of a larger 27 degree-of-freedom (DOF) full-body kinematic model used to optimize other joints as well [58].

Given this model structure, noisy synthetic movement data were generated from a nominal model for which the "true" model parameters were known. Joint parameters for the nominal model along with a nominal motion were derived from in vivo experimental movement data using the optimization methodology described below. Next, three markers were attached to the tibia and calcaneous segments in the model at locations consistent with the experiment, and the 27 model DOFs were moved through their nominal motions. This process created synthetic marker trajectories consistent with the nominal model parameters and motion and also representative of the original experimental data. Finally, numerical noise was added to the synthetic marker trajectories to emulate skin and soft tissue movement artifacts. For each marker coordinate, a sinusoidal noise function was used with uniformly distributed random period, phase, and amplitude (limited to a maximum of ± 1 cm). The values of the sinusoidal parameters were based on previous studies reported in the literature [80,53].

An unconstrained optimization problem with bounds on the design variables was formulated to attempt to recover the known joint parameters from the noisy synthetic marker trajectories. The cost function was

$$\min_P f(P)$$

with
where $p$ is a vector of 12 design variables containing the joint parameters, $q$ is a vector of 27 generalized coordinates for the kinematic model, $c_{ijk}$ is the $i$th coordinate of synthetic marker $j$ at time frame $k$, and $\hat{c}_{ijk}(p,q)$ is the corresponding marker coordinate from the kinematic model. At each time frame, $\hat{c}_{ijk}(p,q)$ was computed from the current model parameters $p$ and an optimized model configuration $q$. A separate Levenberg-Marquardt nonlinear least-squares optimization was performed for each time frame in Eq. (3.20) to determine this optimal configuration. A relative convergence tolerance of $10^{-3}$ was chosen to achieve good accuracy with minimum computational cost. A nested optimization formulation (i.e., minimization occurs in Eqs. (3.19) and (3.20)) was used to decrease the dimensionality of the design space in Eq. (3.19). Equation (3.20) was coded in Matlab and exported as stand-alone C code using the Matlab Compiler (The Mathworks, Natick, MA). The executable read in a file containing the 12 design variables and output a file containing the resulting cost function value. This approach facilitated the use of different optimizers to solve Eq. (3.19).

To investigate the influence of design variable scaling on optimization algorithm performance, two versions of Eq. (3.20) were generated. The first used the original units of centimeters and radians for the position and orientation design variables respectively. Bounds on the design variables were chosen to enclose a physically realistic region around the solution point in design space. Each position design variable was constrained to remain within a cube centered at the midpoint of the medial and lateral malleoli, where the length of each side was equal to the distance between the malleoli (i.e., 11.32 cm). Each orientation design variable was constrained to remain within a circular cone defined
by varying its “true” value by ± 30°. The second version normalized all 12 design variables to be within [-1,1] using

\[ x_{\text{norm}} = \frac{2x - x_{UB} - x_{LB}}{x_{UB} - x_{LB}} \] (3.21)

where \( x_{UB} \) and \( x_{LB} \) denote the upper and lower bounds, respectively, on the design variable vector [81].

Two approaches were used to compare PSO scale sensitivity to that of the other three algorithms. For the first approach, a fixed number of scaled and unscaled runs (10) were performed with each optimization algorithm starting from different random initial seeds, and the sensitivity of the final cost function value to algorithm choice and design variable scaling was evaluated. The stopping condition for PSO and GA runs was 10,000 function evaluations, while SQP and BFGS runs were terminated when a relative convergence tolerance of \( 10^{-5} \) or absolute convergence tolerance of \( 10^{-6} \) was met. For the second approach, a fixed number of function evaluations (10,000) were performed with each algorithm to investigate unscaled versus scaled convergence history. A single random initial guess was used for the PSO and GA algorithms, and each algorithm was terminated once it reached 10,000 function evaluations. Since individual SQP and BFGS runs require much fewer than 10,000 function evaluations, repeated runs were performed with different random initial guesses until the total number of function evaluations exceeded 10,000 at the termination of a run. This approach essentially uses SQP and BFGS as global optimizers, where the separate runs are like individual particles that cannot communicate with each another but have access to local gradient information. Finite difference step size tuning at the start of each run was included in the computation of number of function evaluations. Once the total number of runs required to reach
10,000 function evaluations was known, the lowest cost function value from all runs at each iteration was used to represent the cost over a range of function evaluations equal to the number of runs.

**Results**

For the analytical test problems, our PSO algorithm was more robust than our GA, SQP, and BFGS algorithms (Table 2, top half). PSO converged to the correct global solution nearly 100% of the time on four of the six test problems ($H_1$ and $H_3$ with $n = 4$, 8, and 16). It converged 67% of the time for problem $H_2$ and only 1.5% of the time for problem $H_3$ with $n = 32$. In contrast, none of the other algorithms converged more than 32% of the time on any of the analytical test problems. Though our GA algorithm typically exhibited faster initial convergence than did our PSO algorithm (Figure 7, left column), it leveled off and rarely reached the correct final point in design space within the specified number of function evaluations.

**Table 2** Fraction of successful optimizer runs for the analytical test problems. Top half: Results from the PSO, GA, SQP, and BFGS algorithms used in the present study. Bottom half: Results from the SA, GA, SQP, and DS algorithms used in Soest and Casius 48. The GA and SQP algorithms used in that study were different from the ones used in our study. Successful runs were identified by a final cost function value within of the known optimum value, consistent with [48]

<table>
<thead>
<tr>
<th>Study</th>
<th>Algorithm</th>
<th>$H_1$</th>
<th>$H_2$</th>
<th>(n = 4)</th>
<th>(n = 8)</th>
<th>(n = 16)</th>
<th>(n = 32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Present</td>
<td>PSO</td>
<td>0.972</td>
<td>0.688</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>0.000</td>
<td>0.034</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>SQP</td>
<td>0.09</td>
<td>0.11</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>BFGS</td>
<td>0.00</td>
<td>0.32</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Soest and</td>
<td>SA</td>
<td>1.00</td>
<td>0.027</td>
<td>0.00</td>
<td>0.001</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Casius (2003)</td>
<td>GA</td>
<td>0.990</td>
<td>0.999</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>SQP</td>
<td>0.279</td>
<td>0.810</td>
<td>0.385</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>DS</td>
<td>1.00</td>
<td>0.636</td>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>
Figure 7  Comparison of convergence history results for the analytical test problems. Left column: Results from the PSO, GA, SQP, and BFGS algorithms used in the present study. Right column: Results from the SA, GA, SQP, and DS algorithms used in Soest and Casius [5]. The GA and SQP algorithms used in that study were different from the ones used in our study. (a) Problem $H_1$. The SA results have been updated using corrected data provided by Soest and Casius, since the results in 48 accidentally used a temperature reduction rate of 0.5 rather than the standard value of 0.85 as reported. (b) Problem $H_2$. (c) Problem $H_3$ with $n = 4$. (d) Problem $H_3$ with $n = 32$. Error was computed using the known cost at the global optimum and represents the average of 1000 runs (100 multi-start SQP and BFGS runs in our study) with each algorithm.
In contrast, the SQP and BFGS algorithms were highly sensitive to design variable scaling in the biomechanical test problem. For the ten trials, unscaled and scaled SQP or BFGS runs rarely converged to similar points in design space (note y axis scale in Figure 8) and produced large differences in final cost function value from one trial to the next (Figure 8c and d). Scaling improved the final result in seven out of ten SQP trials and in five of ten BFGS trials. The best unscaled and scaled SQP final cost function values were 255 and 121, respectively, while those of BFGS were 355 and 102 (Table 3). Thus, scaling yielded the best result found with both algorithms. The best SQP and BFGS trials generally produced larger RMS marker distance errors (up to two times worse), orientation parameter errors (up to five times worse), and position parameter errors (up to six times worse) than those found by PSO or GA.
Table 3  Final cost function values and associated marker distance and joint parameter root-mean-square (RMS) errors after 10,000 function evaluations performed by multiple unscaled and scaled PSO, GA, SQP, and BFGS runs. See Figure 9 for corresponding convergence histories.

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>Formulation</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cost Function</td>
</tr>
<tr>
<td>PSOt</td>
<td>Unscaled 69.5</td>
<td>5.44</td>
</tr>
<tr>
<td></td>
<td>Scaled 69.5</td>
<td>5.44</td>
</tr>
<tr>
<td>GA</td>
<td>Unscaled 77.9</td>
<td>5.78</td>
</tr>
<tr>
<td></td>
<td>Scaled 74.0</td>
<td>5.64</td>
</tr>
<tr>
<td>SQP</td>
<td>Unscaled 255</td>
<td>10.4</td>
</tr>
<tr>
<td></td>
<td>Scaled 121</td>
<td>7.21</td>
</tr>
<tr>
<td>BFGS</td>
<td>Unscaled 69.5</td>
<td>5.44</td>
</tr>
<tr>
<td></td>
<td>Scaled 69.5</td>
<td>5.44</td>
</tr>
</tbody>
</table>

Figure 9  Convergence history for unscaled (dark lines) and scaled (gray lines) parallel PSO, GA, SQP, and BFGS runs for the biomechanical test problem. Each algorithm was run terminated after 10,000 function evaluations. Only one unscaled and scaled PSO and GA run were required to reach 10,000 function evaluations, while repeated SQP and BFGS runs were required to reach that number. Separate SQP and BFGS runs were treated like individual particles in a single PSO run for calculating convergence history (see text).
Discussion

This chapter evaluated a recent variation of the PSO algorithm with dynamic inertia and velocity updating as a possible addition to the arsenal of methods that can be applied to difficult biomechanical optimization problems. For all problems investigated, our PSO algorithm with standard algorithm parameters performed better than did three off-the-self optimizers – GA, SQP, and BFGS. For the analytical test problems, PSO robustness was found to be better than that of two other global algorithms but worse than that of a third. For the biomechanical test problem with added numerical noise, PSO was found to be insensitive to design variable scaling while GA was only mildly sensitive and SQP and BFGS highly sensitive. Overall, the results suggest that our PSO algorithm is worth consideration for difficult biomechanical optimization problems, especially those for which design variable scaling may be an issue.

Though our biomechanical optimization involved a system identification problem, PSO may be equally applicable to problems involving forward dynamic, inverse dynamic, inverse static, or image matching analyses. Other global methods such as SA and GA have already been applied successfully to such problems [47,48,62], and there is no reason to believe that PSO would not perform equally well. As with any global optimizer, PSO utilization would be limited by the computational cost of function evaluations given the large number required for a global search.

Our particle swarm implementation may also be applicable to some large-scale biomechanical optimization problems. Outside the biomechanics arena [71,72,82-91], PSO has been used to solve problems on the order of 120 design variables [89-91]. In the present study, our PSO algorithm was unsuccessful on the largest test problem, $H_3$ with $n = 32$ design variables. However, in a recent study, our PSO algorithm successfully solved
the Griewank global test problem with 128 design variables using population sizes ranging from 16 to 128. When the Corana test problem \((H_3)\) was attempted with 128 DVs, the algorithm exhibited worse convergence. Since the Griewank problem possesses a bumpy but continuous search space and the Corana problem a highly discrete search space, our PSO algorithm may work best on global problems with a continuous search space. It is not known how our PSO algorithm would perform on biomechanical problems with several hundred DVs, such as the forward dynamic optimizations of jumping and walking performed with parallel SQP in [44-46].

One advantage of global algorithms such as PSO, GA, and SA is that they often do not require significant algorithm parameter tuning to perform well on difficult problems. The GA used by Soest and Casius in [48] (which is not freely available) required no tuning to perform well on all of these particular analytical test problems. The SA algorithm used by Soest and Casius in [48] required tuning of two parameters to improve algorithm robustness significantly on those problems. Our PSO algorithm (which is freely available) required tuning of one parameter \((w_d)\), which was increased from 1.0 to 1.5) to produce 100% success on the two problems where it had significant failures. For the biomechanical test problem, our PSO algorithm required no tuning, and only the population size of our GA algorithm required tuning to improve convergence speed. Neither algorithm was sensitive to the two sources of noise present in the problem - noise added to the synthetic marker trajectories, and noise due to a somewhat loose convergence tolerance in the Levenberg-Marquardt sub-optimizations. Thus, for many global algorithm implementations, poor performance on a particular problem can be rectified by minor tuning of a small number of algorithm parameters.
Figure 10  Sensitivity of gradient calculations to selected finite difference step size for one design variable. Forward and central differencing were evaluated using relative convergence tolerances of $10^{-3}$ and $10^{-6}$ for the nonlinear least squares sub-optimizations performed during cost function evaluation (see Eq. (3.20)).

In contrast, gradient-based algorithms such as SQP and BFGS can require a significant amount of tuning even to begin to approach global optimizer results on some problems. For the biomechanical test problem, our SQP and BFGS algorithms were highly tuned by scaling the design variables and determining the optimal FDSS for each design variable separately. FDSS tuning was especially critical due to the two sources of noise noted above. When forward and central difference gradient results were compared for one of the design variables using two different Levenberg-Marquardt relative convergence tolerances ($10^{-3}$ and $10^{-6}$), a "sweet spot" was found near a step size of $10^{-2}$ (Figure 10). Outside of that "sweet spot," which was automatically identified and used in generating our SQP and BFGS results, forward and central difference gradient results diverged quickly when the looser tolerance was used. Since most users of gradient-based optimization algorithms do not scale the design variables or tune the FDSS for each design variable separately, and many do not perform multiple runs, our SQP and BFGS
results for the biomechanical test problem represent best-case rather than typical results. For this particular problem, an off-the-shelf global algorithm such as PSO or GA is preferable due to the significant reduction in effort required to obtain repeatable and reliable solutions.

Another advantage of PSO and GA algorithms is the ease with which they can be parallelized [48,76] and their resulting high parallel efficiency. For our PSO algorithm, Schutte et al. [76] recently reported near ideal parallel efficiency for up to 32 processors. Soest and Casius [48] reported near ideal parallel efficiency for their GA algorithm with up to 40 processors. Though SA has historically been considered more difficult to parallelize [92], Higginson et al. [93] recently developed a new parallel SA implementation and demonstrated near ideal parallel efficiency for up to 32 processors. In contrast, Koh et al. [77] reported poor SQP parallel efficiency for up to 12 processors due to the sequential nature of the line search portion of the algorithm.

The caveat for these parallel efficiency results is that the time required per function evaluation was approximately constant and the computational nodes were homogeneous. As shown in [76], when function evaluations take different amounts of time, parallel efficiency of our PSO algorithm (and any other synchronous parallel algorithm, including GA, SA, SQP, and BFGS) will degrade with increasing number of processors. Synchronization between individuals in the population or between individual gradient calculations requires slave computational nodes that have completed their function evaluations to sit idle until all nodes have returned their results to the master node. Consequently, the slowest computational node (whether loaded by other users, performing the slowest function evaluation, or possessing the slowest processor in a
heterogeneous environment) will dictate the overall time for each parallel iteration. An asynchronous PSO implementation with load balancing, where the global best-found position is updated continuously as each particle completes a function evaluation, could address this limitation. However, the extent to which convergence characteristics and scale independence would be affected is not yet known.

To put the results of our study into proper perspective, one must remember that optimization algorithm robustness can be influenced heavily by algorithm implementation details, and no single optimization algorithm will work for all problems. For two of the analytical test problems ($H_2$ and $H_3$ with $n = 4$), other studies have reported PSO results using formulations that did not include dynamic inertia and velocity updating. Comparisons are difficult given differences in the maximum number of function evaluations and number of particles, but in general, algorithm modifications were (not surprisingly) found to influence algorithm convergence characteristics [94-96]. For our GA and SQP algorithms, results for the analytical test problems were very different from those obtained by Soest and Casius in [48] using different GA and SQP implementations. With seven mutation and four crossover operators, the GA algorithm used by Soest and Casius in [48] was obviously much more complex than the one used here. In contrast, both SQP algorithms were highly-developed commercial implementations. In contrast, poor performance by a gradient-based algorithm can be difficult to correct even with design variable scaling and careful tuning of the FDSS. These findings indicate that specific algorithm implementations, rather than general classes of algorithms, must be evaluated to reach any conclusions about algorithm robustness and performance on a particular problem.
Conclusions

In summary, the PSO algorithm with dynamic inertia and velocity updating provides another option for difficult biomechanical optimization problems with the added benefit of being scale independent. There are few algorithm-specific parameters to adjust, and standard recommended settings work well for most problems [70,94]. In biomechanical optimization problems, noise, multiple local minima, and design variables of different scale can limit the reliability of gradient-based algorithms. The PSO algorithm presented here provides a simple-to-use off-the-shelf alternative for consideration in such cases.

The algorithm’s main drawback is the high cost in terms of function evaluations because of slow convergence in the final stages of the optimization, a common trait among global search algorithms. The time requirements associated with the high computational cost may be circumvented by utilizing the parallelism inherent in the swarm algorithm. The development of such a parallel PSO algorithm will be detailed in the next chapter.
CHAPTER 4
PARALLELISM BY EXPLOITING POPULATION-BASED ALGORITHM STRUCTURES

Overview

The structures of population based optimizers such as Genetic Algorithms and the Particle Swarm may be exploited in order to enable these algorithms to utilize concurrent processing. These algorithms require a set of fitness values for the population or swarm of individuals for each iteration during the search. The fitness of each individual is independently evaluated, and may be assigned to separate computational nodes. The development of such a parallel computational infrastructure is detailed in this chapter and applied to a set of large-scale analytical problems and a biomechanical system identification problem for the purpose of quantifying its efficiency. The parallelization of the PSO is achieved with a master-slave, coarse-grained implementation where slave computational nodes are associated with individual particle search trajectories and assigned their fitness evaluations. Greatly enhanced computation throughput is demonstrated using this infrastructure, with efficiencies of 95% observed for load balanced conditions. Numerical example problems with large load imbalances yield poor performance, which decreases in a linear fashion as additional nodes are added. This infrastructure is based on a two-level approach with flexibility in terms of where the search effort can be concentrated. For the two problems presented, the global search effort is applied in the upper level.
This work was done in collaboration with Jeff Reinbolt, who created the biomechanical kinematic analysis software [58] and evaluated the quality of the solutions found by the parallel PSO. The work presented in this chapter was also published in [58,76,129].

**Introduction**

Present day engineering optimization problems often impose large computational demands, resulting in long solution times even on a modern high-end processor. To obtain enhanced computational throughput and global search capability, we detail the coarse-grained parallelization of an increasingly popular global search method, the particle swarm optimization (PSO) algorithm. Parallel PSO performance was evaluated using two categories of optimization problems possessing multiple local minima —large-scale analytical test problems with computationally cheap function evaluations and medium-scale biomechanical system identification problems with computationally expensive function evaluations. For load-balanced analytical test problems formulated using 128 design variables, speedup was close to ideal and parallel efficiency above 95% for up to 32 nodes on a Beowulf cluster. In contrast, for load-imbalanced biomechanical system identification problems with 12 design variables, speedup plateaued and parallel efficiency decreased almost linearly with increasing number of nodes. The primary factor affecting parallel performance was the synchronization requirement of the parallel algorithm, which dictated that each iteration must wait for completion of the slowest fitness evaluation. When the analytical problems were solved using a fixed number of swarm iterations, a single population of 128 particles produced a better convergence rate than did multiple independent runs performed using sub-populations (8 runs with 16 particles, 4 runs with 32 particles, or 2 runs with 64 particles). These results suggest that
(1) parallel PSO exhibits excellent parallel performance under load-balanced conditions, (2) an asynchronous implementation would be valuable for real-life problems subject to load imbalance, and (3) larger population sizes should be considered when multiple processors are available.

Numerical optimization has been widely used in engineering to solve a variety of NP-complete problems in areas such as structural optimization, neural network training, control system analysis and design, and layout and scheduling problems. In these and other engineering disciplines, two major obstacles limiting the solution efficiency are frequently encountered. First, even medium-scale problems can be computationally demanding due to costly fitness evaluations. Second, engineering optimization problems are often plagued by multiple local optima, requiring the use of global search methods such as population-based algorithms to deliver reliable results. Fortunately, recent advances in microprocessor and network technology have led to increased availability of low cost computational power through clusters of low to medium performance computers. To take advantage of these advances, communication layers such as MPI [3, 5] and PVM [97] have been used to develop parallel optimization algorithms, the most popular being gradient-based, genetic (GA), and simulated annealing (SA) algorithms [48,98,99]. In biomechanical optimizations of human movement, for example, parallelization has allowed problems requiring days or weeks of computation on a single-processor computer to be solved in a matter of hours on a multi-processor machine [98]. The particle swarm optimization (PSO) algorithm is a recent addition to the list of global search methods [100]. This derivative-free method is particularly suited to continuous variable problems and has received increasing attention in the optimization community. It
has been successfully applied to large-scale problems [69,100,101] in several engineering disciplines and, being a population-based approach, is readily parallelizable. It has few algorithm parameters, and generic settings for these parameters work well on most problems [70,94]. In this study, we present a parallel PSO algorithm for application to computationally demanding optimization problems. The algorithm’s enhanced throughput due to parallelization and improved convergence due to increased population size are evaluated using large-scale analytical test problems and medium-scale biomechanical system identification problems. Both types of problems possess multiple local minima. The analytical test problems utilize 128 design variables to create a tortuous design space but with computationally cheap fitness evaluations. In contrast, the biomechanical system identification problems utilize only 12 design variables but each fitness evaluation is much more costly computationally. These two categories of problems provide a range of load balance conditions for evaluating the parallel formulation.

**Serial Particle Swarm Algorithm**

Particle swarm optimization was introduced in 1995 by Kennedy and Eberhart [66]. Although several modifications to the original swarm algorithm have been made to improve performance [68,102-105] and adapt it to specific types of problems [69,106,107], a parallel version has not been previously implemented. The following is a brief introduction to the operation of the PSO algorithm. Consider a swarm of \( p \) particles, with each particle’s position representing a possible solution point in the design problem space \( D \). For each particle \( i \), Kennedy and Eberhart proposed that its position \( x_i \) be updated in the following manner:
with a pseudo-velocity \( v_i \) calculated as follows:

\[
    v_{k+1}^i = w_k v_k^i + c_1 r_1^i \left( p_k^i - x_k^i \right) + c_2 r_2 \left( g_k - x_k^i \right)
\]

(4.2)

Here, subscript \( k \) indicates a (unit) pseudo-time increment, \( p_i^k \) represents the best ever position of particle \( i \) at time \( k \) (the cognitive contribution to the pseudo-velocity vector \( v_i^{k+1} \)), and \( p^g_k \) represents the global best position in the swarm at time \( k \) (social contribution). \( r_1 \) and \( r_2 \) represent uniform random numbers between 0 and 1. To allow the product \( c_1 r_1 \) or \( c_2 r_2 \) to have a mean of 1, Kennedy and Eberhart proposed that the cognitive and social scaling parameters \( c_1 \) and \( c_2 \) be selected such that \( c_1 = c_2 = 2 \). The result of using these proposed values is that the particles overshoot the target half the time, thereby maintaining separation within the group and allowing for a greater area to be searched than if no overshoot occurred. A modification by Fourie and Groenwold [69] on the original PSO algorithm 66 allows transition to a more refined search as the optimization progresses. This operator reduces the maximum allowable velocity \( v_{max}^k \) and particle inertia \( w^k \) in a dynamic manner, as dictated by the dynamic reduction parameters \( \kappa_d, w_d \). For the sake of brevity, further details of this operator are omitted, but a detailed description can be found in References [69,70]. The serial PSO algorithm as it would typically be implemented on a single CPU computer is described below, where \( p \) is the total number of particles in the swarm. The best ever fitness value of a particle at design coordinates \( p_i^k \) is denoted by \( f_{i}^{best} \) and the best ever fitness value of the overall swarm at coordinates \( p^g_k \) by \( f^g_{best} \). At time step \( k = 0 \), the particle velocities \( v_1^0 \) are initialized to values within the limits \( 0 \leq v_0 \leq v_{max}^0 \). The vector \( v_{max}^0 \) is calculated as a fraction of the distance between the upper and lower bounds \( v_{max}^0 = \zeta(x_{UB} - x_{LB}) \) [69], with \( \zeta = 0.5 \).

With this background, the PSO algorithm flow can be described as follows:
6. Initialize

a. Set constants $\kappa$, $c_1$, $c_2$, $k_{\text{max}}$, $v_{\text{max}}^i$, $w_0$, $v_d$, $w_d$, and $d$

b. Initialize dynamic maximum velocity $v_{\text{max}}^i$ and inertia $w_k$

c. Set counters $k = 0$, $t = 0$, $i = 1$. Set random number seed.

d. Randomly initialize particle positions $x_0^i \in D$ in $\mathcal{R}$ for $i = 1, \ldots, p$

e. Randomly initialize particle velocities $0 \leq v_0^i \leq v_{\text{max}}^i$ for $i = 1, \ldots, p$

f. Evaluate cost function values $f_0^i$ using design space coordinates $x_0^i$ for $i = 1, \ldots, p$

g. Set $f_{\text{best}}^i = f_0^i$ and $p_0^i = x_0^i$ for $i = 1, \ldots, p$

h. Set $f_{\text{best}}^g$ to best $f_{\text{best}}^i$ and $g_0$ to corresponding $x_0^i$

7. Optimize

a. Update particle velocity vectors $v_{k+1}^i$ using Eq.(4.2)

b. If $v_{k+1}^i > v_{\text{max}}^i$ for any component, then set that component to its maximum allowable value

c. Update particle position vectors $x_{k+1}^i$ using Eq. (4.1)

d. Evaluate cost function values $f_{k+1}^i$ using design space coordinates $x_{k+1}^i$ for $i = 1, \ldots, p$

e. If $f_{k+1}^i \leq f_{\text{best}}^i$, then $f_{\text{best}}^i = f_{k+1}^i$, $p_{k+1}^i = x_{k+1}^i$ for $i = 1, \ldots, p$

f. If $f_{k+1}^i \leq f_{\text{best}}^g$, then $f_{\text{best}}^g = f_{k+1}^i$, $g_{k+1}^i = x_{k+1}^i$ for $i = 1, \ldots, p$

g. If $f_{\text{best}}^g$ was improved in (e), then reset $t = 0$. Else increment $t$. If $k > k_{\text{max}}$ go to 3

h. If $t = d$, then multiply $w_{k+1}$ by $(1 - w_d)$ and $v_{\text{max}}^i$ by $(1 - v_d)$

i. If maximum number of function evaluations is exceeded, then go to 3

j. Increment $i$. If $i > p$ then increment $k$, and set $i = 1$
k. Go to 2(a).

8. Report results

9. Terminate

The above logic is illustrated as a flow diagram in Figure 11 without detailing the working of the dynamic reduction parameters. Problem independent stopping conditions based on convergence tests are difficult to define for global optimizers. Consequently, we typically use a fixed number of fitness evaluations or swarm iterations as a stopping criteria.

**Parallel Particle Swarm Algorithm**

The following issues had to be addressed in order to create a parallel PSO algorithm.

**Concurrent Operation and Scalability**

The algorithm should operate in such a fashion that it can be easily decomposed for parallel operation on a multi-processor machine. Furthermore, it is highly desirable that it be scalable. Scalability implies that the nature of the algorithm should not place a limit on the number of computational nodes that can be utilized, thereby permitting full use of available computational resources. An example of an algorithm with limited scalability is a parallel implementation of a gradient-based optimizer. This algorithm is decomposed by distributing the workload of the derivative calculations for a single point in design space among multiple processors. The upper limit on concurrent operations using this approach is therefore set by the number of design variables in the problem. On the other hand, population-based methods such as the GA and PSO are better suited to parallel computing. Here the population of individuals representing designs can be increased or decreased according to the availability and speed of processors. Any additional agents in the population will allow for a higher fidelity search in the design space, lowering
susceptibility to entrapment in local minima. However, this comes at the expense of additional fitness evaluations.

Figure 11  Serial implementation of PSO algorithm. To avoid complicating the diagram, we have omitted velocity/inertia reduction operations.

**Asynchronous vs. Synchronous Implementation**

The original PSO algorithm was implemented with a synchronized scheme for updating the best ‘remembered’ individual and group fitness values $f_i^k$ and $f_g^k$. 
respectively, and their associated positions $p^i_k$ and $p^g_k$. This approach entails performing the fitness evaluations for the entire swarm before updating the best fitness values. Subsequent experimentation revealed that improved convergence rates can be obtained by updating the $f^i_k$ and $f^g_k$ values and their positions after each individual fitness evaluation (i.e. in an asynchronous fashion) [70,94].

It is speculated that because the updating occurs immediately after each fitness evaluation, the swarm reacts more quickly to an improvement in the best-found fitness value. With the parallel implementation, however, this asynchronous improvement on the swarm is lost since fitness evaluations are performed concurrently. The parallel algorithm requires updating $f^i_k$ and $f^g_k$ for the entire swarm after all fitness evaluations have been performed, as in the original particle swarm formulation. Consequently, the swarm will react more slowly to changes of the best fitness value ‘position’ in the design space. This behavior produces an unavoidable performance loss in terms of convergence rate compared to the asynchronous implementation and can be considered part of the overhead associated with parallelization.

**Coherence**

Parallelization should have no adverse affect on algorithm operation. Calculations sensitive to program order should appear to have occurred in exactly the same order as in the serial synchronous formulation, leading to the exact same final answer. In the serial PSO algorithm the fitness evaluations form the bulk of the computational effort for the optimization and can be performed independently. For our parallel implementation, we therefore chose a coarse decomposition scheme where the algorithm performs only the fitness evaluations concurrently on a parallel machine. Step 2 of the particle swarm optimization algorithm was modified accordingly to operate in a parallel manner:
2) Optimize

a) Update particle velocity vectors $v'_{k+1}$ using Eq.(4.2)

b) If $v'_{k+1} > v^\text{max}_{k+1}$ for any component, then set that component to its maximum allowable value

c) Update particle position vectors $x'_{k+1}$ using Eq. (4.1)

d) Concurrently evaluate fitness values $f^i_{k+1}$ using design space co-ordinates $x^i_{k+1}$ for $i = 1,\ldots, p$

e) If $f^i_{k+1} \leq f^\text{best}^i$, then $f^\text{best} = f^i_{k+1}$, $p^i_{k+1} = x^i_{k+1}$ for $i = 1,\ldots, p$

f) If $f^i_{k+1} \leq f^g_{\text{best}}$, then $f^\text{best} = f^i_{k+1}$, $g^i_{k+1} = x^i_{k+1}$ for $i = 1,\ldots, p$

g) If $f^g_{\text{best}}$ was improved in (e), then reset $t = 0$. Else increment $t$. If $k > k_{\text{max}}$ go to 3

h) If $t = d$, then multiply $w_{k+1}$ by $(1 - w_d)$ and $v^\text{max}_{k+1}$ by $(1 - v_d)$

i) If maximum number of function evaluations is exceeded, then go to 3

j) Increment $k$

k) Go to 2(a).

The parallel PSO algorithm is represented by the flow diagram in Figure 12.

Network Communication

In a parallel computational environment, the main performance bottleneck is often the communication latency between processors. This issue is especially relevant to large clusters of computers where the use of high performance network interfaces are limited due to their high cost. To keep communication between different computational nodes at a minimum, we use fitness evaluation tasks as the level of granularity for the parallel software. As previously mentioned each of these evaluations can be performed.
independently and requires no communication aside from receiving design space coordinates to be evaluated and reporting the fitness value at the end of the analysis.

Figure 12 Parallel implementation of the PSO algorithm. We have again omitted velocity/inertia reduction operations to avoid complicating the diagram.
The optimization infrastructure is organized into a coordinating node and several computational nodes. PSO algorithm functions and task orchestration are performed by the coordinating node, which assigns the design co-ordinates to be evaluated, in parallel, to the computational nodes. With this approach, no communication is required between computational nodes as individual fitness evaluations are independent of each other. The only necessary communication is between the coordinating node and the computational nodes and encompasses passing the following information:

1) Several distinct design variable configuration vectors assigned by coordinating node to slave nodes for fitness evaluation.
2) Fitness values reported from slave nodes to coordinating node.
3) Synchronization signals to maintain program coherence.
4) Termination signals from coordinating node to slave nodes on completion of analysis to stop the program cleanly.

The parallel PSO scheme and required communication layer were implemented in ANSI C on a Linux operating system using the message passing interface (MPI) libraries.

**Synchronization and Implementation**

From the parallel PSO algorithm, it is clear that some means of synchronization is required to ensure that all of the particle fitness evaluations have been completed and results reported before the velocity and position calculations can be executed (steps 2a and 2b). Synchronization is done using a barrier function in the MPI communication library which temporarily stops the coordinating node from proceeding with the next swarm iteration until all of the computational nodes have responded with a fitness value. Because of this approach, the time required to perform a single parallel swarm fitness evaluation will be dictated by the slowest fitness evaluation in the swarm. Two networked clusters of computers were used to obtain the numerical results. The first
cluster was used to solve the analytical test problems and comprised 40 1.33 GHz Athlon PCs located in the High-performance Computing and Simulation (HCS) Research Laboratory at the University of Florida. The second group was used to solve the biomechanical system identification problems and consisted of 32 2.40 GHz Intel PCs located in the HCS Research Laboratory at Florida State University. In both locations, 100 Mbps switched networks were utilized for connecting nodes.

Sample Optimization Problems

Analytical Test Problems

Two well-known analytical test problems were used to evaluate parallel PSO algorithm performance on large-scale problems with multiple local minima (see Appendix A for mathematical description of both problems). The first was a test function (Figure 13 (a)) introduced by Griewank [108] which superimposes a high-frequency sine wave on a multi-dimensional parabola. In contrast, the second problem used the Corana test function [109] which exhibits discrete jumps throughout the design space (Figure 13(b)). For both problems, the number of local minima increases exponentially with the number of design variables. To investigate large-scale optimization issues, we formulated both problems using 128 design variables. Since fitness evaluations are extremely fast for these test problems, a delay of approximately half a second was built into each fitness evaluation so that total computation time would not be swamped by communication time. Since parallelization opens up the possibility of utilizing large numbers of processors, we used the analytical test problems to investigate how convergence rate and final solution are affected by the number of particles employed in a parallel PSO run. To ensure that all swarms were given equally ‘fair’ starting positions, we generated a pool of 128 initial
positions using the Latin Hypercube Sampler (LHS). Particle positions selected with this scheme will be distributed uniformly throughout the design space [110].

This initial pool of 128 particles was divided into the following sub-swarms: one swarm of 128 particles, two swarms of 64 particles, four swarms of 32 particles, and eight swarms of 16 particles. Each sub-swarm was used independently to solve the two analytical test problems. This approach allowed us to investigate whether it is more efficient to perform multiple parallel optimizations with smaller population sizes or one parallel optimization with a larger population size given a sufficient number of processors. To obtain comparisons for convergence speed, we allowed all PSO runs to complete 10,000 iterations before the search was terminated. This number of iterations corresponded to between 160,000 and 1,280,000 fitness evaluations depending on the number of particles employed in the swarm.

**Biomechanical System Identification problems**

In addition to the analytical test problems, medium-scale biomechanical system identification problems were used to evaluate parallel PSO performance under more realistic conditions. These problems were variations of a general problem that attempts to find joint parameters (i.e. positions and orientations of joint axes) that match a kinematic ankle model to experimental surface marker data [56]. The data are collected with an optoelectronic system that uses multiple cameras to record the positions of external markers placed on the body segments. To permit measurement of three-dimensional motion, we attach three non-collinear markers to the foot and lower leg. The recordings are processed to obtain marker trajectories in a laboratory-fixed co-ordinate system [111, 112]. The general problem possesses 12 design variables and requires approximately 1 minute for each fitness evaluation. Thus, while the problem is only medium-scale in
terms of number of design variables, it is still computationally costly due to the time required for each fitness evaluation.

Figure 13 Surface plots of the (a) Griewank and (b) Corana analytical test problems showing the presence of multiple local minima. For both plots, 126 design variables were fixed at their optimal values and the remaining 2 design variables varied in a small region about the global minimum.

The first step in the system identification procedure is to formulate a parametric ankle joint model that can emulate a patient’s movement by possessing sufficient degrees of freedom. For the purpose of this paper, we approximate the talocrural and subtalar
joints as simple 1 degree of freedom revolute joints. The resulting ankle joint model (Figure 6) contains 12 adjustable parameters that define its kinematic structure [56]. The model also has a set of virtual markers fixed to the limb segments in positions corresponding to the locations of real markers on the subject. The linkage parameters are then adjusted via optimization until markers on the model follow the measured marker trajectories as closely as possible. To quantify how closely the kinematic model with specified parameter values can follow measured marker trajectories, we define a cumulative marker error $e$ as follows:

$$ e = \sum_{j=1}^{n} \sum_{i=1}^{m} \Delta_{i,j}^2 $$

(4.3)

where

$$ \Delta_{i,j}^2 = \Delta x_{i,j}^2 + \Delta y_{i,j}^2 + \Delta z_{i,j}^2 $$

(4.4)

where $\Delta x_{i,j}, \Delta y_{i,j}$ and $\Delta z_{i,j}$ are the spatial displacement errors for marker $i$ at time frame $j$ in the $x, y, z$ directions as measured in the laboratory-fixed coordinate system, $n = 50$ is the number of time frames, and $m = 6$ (3 on the lower leg and 3 on the foot) is the number of markers. These errors are calculated between the experimental marker locations on the human subject and the virtual marker locations on the kinematic model.

For each time frame, a non-linear least squares sub-optimization is performed to determine the joint angles that minimize $\Delta_{i,j}^2$ given the current set of model parameters. The first sub-optimization is started from an initial guess of zero for all joint angles. The sub-optimization for each subsequent time frame is started with the solution from the previous time frame to speed convergence. By performing a separate sub-optimization for each time frame and then calculating the sum of the squares of the marker co-ordinate errors, we obtain an estimate of how well the model fits the data for all time frames.
included in the analysis. By varying the model parameters and repeating the sub-
optimization process, the parallel PSO algorithm finds the best set of model parameters
that minimize e over all time frames.

For numerical testing, three variations of this general problem were analyzed as
described below. In all cases the number of particles used by the parallel PSO algorithm
was set to a recommended value of 20 [94].

1) **Synthetic data without numerical noise**: Synthetic (i.e. computer generated)
data without numerical noise were generated by simulating marker movements
using a lower body kinematic model with virtual markers. The synthetic motion
was based on an experimentally measured ankle motion (see 3 below). The
kinematic model used anatomically realistic joint positions and orientations. Since
the joint parameters associated with the synthetic data were known, his
optimization was used to verify that the parallel PSO algorithm could accurately
recover the original model.

2) **Synthetic data with numerical noise**: Numerical noise was superimposed on
each synthetic marker coordinate trajectory to emulate the effect of marker
displacements caused by skin movement artifacts [53]. A previously published
noise model requiring three random parameters was used to generate a
perturbation $N$ in each marker coordinate [80]:

$$N = A \sin(\omega t + \phi)$$ (4.5)

where $A$ is the amplitude, $\omega$ the frequency, and $\phi$ the phase angle of the noise.
These noise parameters were treated as uniform random variables within the
bounds $0 \leq A \leq 1$ cm, $0 \leq \omega \leq 25$ rad/s, and $0 \leq \phi \leq 2$ (obtained from [80]).

3) **Experimental data**: Experimental marker trajectory data were obtained by
processing three-dimensional recordings from a subject performing movements
with reflective markers attached to the foot and lower leg as previously described.
Institutional review board approval was obtained for the experiments and data
analysis, and the subject gave informed consent prior to participation. Marker
positions were reported in a laboratory-fixed coordinate system.

**Speedup and Parallel Efficiency**

Parallel performance for both classes of problems was quantified by calculating
speedup and parallel efficiency for different numbers of processors. Speedup is the ratio
of sequential execution time to parallel execution time and ideally should equal the
number of processors. Parallel efficiency is the ratio of speedup to number of processors and ideally should equal 100%. For the analytical test problems, only the Corana problem was run since the half second delay added to both problems makes their parallel performance identical. For the biomechanical system identification problems, only the synthetic data with numerical noise case was reported since experimentation with the other two cases produced similar parallel performance.

Figure 14  Average fitness convergence histories for the (a) Griewank and (b) Corana analytical test problems for swarm sizes of 16, 32, 64, and 128 particles and 10000 swarm iterations. Triangles indicate the location on each curve where 160 000 fitness evaluations were completed.
The number of particles and nodes used for each parallel evaluation was selected based on the requirements of the problem. The Corana problem with 128 design variables was solved using 32 particles and 1, 2, 4, 8, 16, and 32 nodes. The biomechanical problem with 12 design variables was solved using 20 particles and 1, 2, 5, 10, and 20 nodes. Both problems were allowed to run until 1000 fitness evaluations were completed.

**Numerical Results**

Convergence rates for the two analytical test problems differed significantly with changes in swarm size. For the Griewank problem (Figure 14(a)), individual PSO runs converged to within 1e-6 of the global minimum after 10 000 optimizer iterations, regardless of the swarm size. Run-to-run variations in final fitness value (not shown) for a fixed swarm size were small compared to variations between swarm sizes. For example, no runs with 16 particles produced a better final fitness value than any of the runs with 32 particles, and similarly for the 16–32, 32–64, and 64–128 combinations. When number of fitness evaluations was considered instead of number of swarm iterations, runs with a smaller swarm size tended to converge more quickly than did runs with a larger swarm size (see triangles in Figure 14). However, two of the eight runs with the smallest number of particles failed to show continued improvement near the maximum number of iterations, indicating possible entrapment in a local minimum. Similar results were found for the Corana problem (Figure 14(b)) with two exceptions. First, the optimizer was unable obtain the global minimum for any swarm size within the specified number of iterations (Figure 14(b)), and second, overlapping in results between different swarm sizes was observed. For example, some 16 particle results were better than 32 particles results, and similarly for the other neighboring combinations. On average, however, a larger swarm size tended to produce better results for both problems.
Table 4 Parallel PSO results for the biomechanical system identification problem using synthetic marker trajectories without and with numerical noise. Optimizations on synthetic with noise and without noise were with 20 particles and were terminated after 40000 fitness evaluations.

<table>
<thead>
<tr>
<th>Model parameter</th>
<th>Upper bound</th>
<th>Lower bound</th>
<th>Synthetic solution</th>
<th>Synthetic data Without noise</th>
<th>Synthetic data With noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁ (deg)</td>
<td>48.67</td>
<td>-11.63</td>
<td>18.37</td>
<td>18.36</td>
<td>15.13</td>
</tr>
<tr>
<td>p₂ (deg)</td>
<td>30.00</td>
<td>-30.00</td>
<td>0.00</td>
<td>-0.01</td>
<td>8.01</td>
</tr>
<tr>
<td>p₃ (deg)</td>
<td>70.23</td>
<td>10.23</td>
<td>40.23</td>
<td>40.26</td>
<td>32.97</td>
</tr>
<tr>
<td>p₄ (deg)</td>
<td>53.00</td>
<td>-7.00</td>
<td>23.00</td>
<td>23.03</td>
<td>23.12</td>
</tr>
<tr>
<td>p₅ (deg)</td>
<td>72.00</td>
<td>12.00</td>
<td>42.00</td>
<td>42.00</td>
<td>42.04</td>
</tr>
<tr>
<td>p₆ (cm)</td>
<td>6.27</td>
<td>-6.27</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.39</td>
</tr>
<tr>
<td>p₇ (cm)</td>
<td>-33.70</td>
<td>-46.24</td>
<td>-39.97</td>
<td>-39.97</td>
<td>-39.61</td>
</tr>
<tr>
<td>p₈ (cm)</td>
<td>6.27</td>
<td>-6.27</td>
<td>0.00</td>
<td>-0.00</td>
<td>0.76</td>
</tr>
<tr>
<td>p₉ (cm)</td>
<td>0.00</td>
<td>-6.27</td>
<td>-1.00</td>
<td>-1.00</td>
<td>-2.82</td>
</tr>
<tr>
<td>p₁₀ (cm)</td>
<td>15.27</td>
<td>2.72</td>
<td>9.00</td>
<td>9.00</td>
<td>10.21</td>
</tr>
<tr>
<td>p₁₁ (cm)</td>
<td>10.42</td>
<td>-2.12</td>
<td>4.15</td>
<td>4.15</td>
<td>3.03</td>
</tr>
<tr>
<td>p₁₂ (cm)</td>
<td>6.89</td>
<td>-5.65</td>
<td>0.62</td>
<td>0.62</td>
<td>-0.19</td>
</tr>
</tbody>
</table>

The parallel PSO algorithm found ankle joint parameters consistent with the known solution or results in the literature [61-63]. The algorithm had no difficulty recovering the original parameters from the synthetic data set without noise (Table 4), producing a final cumulative error $e$ on the order of $10^{-13}$. The original model was recovered with mean orientation errors less than $0.05^\circ$ and mean position errors less than $0.008\text{cm}$.

Furthermore, the parallel implementation produced identical fitness and parameter histories as did a synchronous serial implementation. For the synthetic data set with superimposed noise, a RMS marker distance error of 0.568cm was found, which is on the order of the imposed numerical noise with maximum amplitude of 1cm. For the experimental data set, the RMS marker distance error was 0.394cm (Table 5), comparable to the error for the synthetic data with noise. Convergence characteristics were similar for the three data sets considered in this study. The initial convergence rate
was quite high (Figure 15(a)), where after it slowed when the approximate location of the global minimum was found.

Table 5  Parallel PSO results for the biomechanical system identification problem using synthetic marker trajectories without and with numerical noise.

<table>
<thead>
<tr>
<th></th>
<th>Synthetic Data</th>
<th>Experimental data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Without noise</td>
<td>With noise</td>
</tr>
<tr>
<td>Marker distances (cm)</td>
<td>3.58E-04</td>
<td>0.568</td>
</tr>
<tr>
<td>Orientation parameters (deg)</td>
<td>1.85E-02</td>
<td>5.010</td>
</tr>
<tr>
<td>Position parameters (cm)</td>
<td>4.95E-04</td>
<td>1.000</td>
</tr>
</tbody>
</table>

As the solution process proceeded, the optimizer traded off increases in RMS joint orientation error (Figure 15(b)) for decreases in RMS joint position error (Figure 15(c)) to achieve further minor reductions in the fitness value.

The analytical and biomechanical problems exhibited different parallel performance characteristics. The analytical problem demonstrated almost perfectly linear speedup (Figure 16(a), squares) resulting in parallel efficiencies above 95% for up to 32 nodes (Figure 16(b), squares). In contrast, the biomechanical problem exhibited speedup results that plateaued as the number of nodes was increased (Figure 16(a), circles), producing parallel efficiencies that decreased almost linearly with increasing number of nodes (Figure 16(b), circles). Each additional node produced roughly a 3% reduction in parallel efficiency.

**Discussion**

This study presented a parallel implementation of the particle swarm global optimizer. The implementation was evaluated using analytical test problems and biomechanical system identification problems. Speedup and parallel efficiency results were excellent when each fitness evaluation took the same amount of time.
Figure 15  Fitness convergence and parameter error plots for the biomechanical system identification problem using synthetic data with noise
Figure 16  (a) Speedup and (b) parallel efficiency for the analytical and biomechanical optimization problems.

For problems with large numbers of design variables and multiple local minima, maximizing the number of particles produced better results than repeated runs with fewer particles. Overall, parallel PSO makes efficient use of computational resources and provides a new option for computationally demanding engineering optimization problems. The agreement between optimized and known orientation parameters $p_1$–$p_4$ for the biomechanical problem using synthetic data with noise was poorer than initially expected. This finding was the direct result of the sensitivity of orientation calculations to errors in marker positions caused by the injected numerical noise. Because of the close proximity of the markers to each other, even relatively small amplitude numerical noise
in marker positions can result in large fluctuations in the best-fit joint orientations. While more time frames could be used to offset the effects of noise, this approach would increase the cost of each fitness evaluation due to an increased number of sub-optimizations. Nonetheless, the fitness value for the optimized parameters was lower than that for the parameters used to generate the original noiseless synthetic data.

Though the biomechanical optimization problems only involved 12 design variables, multiple local minima existed when numerical or experimental noise was present. When the noisy synthetic data set was analyzed with a gradient-based optimizer using 20 random starting points, the optimizer consistently found distinct solutions, indicating a large number of local minima. Similar observations were made for a smaller number of gradient-based runs performed on the experimental data set. To evaluate the parallel PSOs ability to avoid entrapment in these local minima, we performed 10 additional runs with the algorithm. All 10 runs converged to the same solution, which was better than any of the solutions found by gradient-based runs.

Differences in parallel PSO performance between the analytical test problem and the biomechanical system identification problem can be explained by load balancing issues. The half second delay added to the analytical test problem made all fitness evaluations take approximately the same amount of time and substantially less time than communication tasks. Consequently, load imbalances were avoided and little degradation in parallel performance was observed with increasing number of processors. In contrast, for the biomechanical system identification problem, the time required to complete the 50 sub-optimizations was sensitive to the selected point in design space, thereby producing load imbalances. As the number of processors increased, so did the likelihood that at least
one fitness evaluation would take much longer than the others. Due to the synchronization requirement of the current parallel implementation, the resulting load imbalance caused by even one slow fitness evaluation was sufficient to degrade parallel performance rapidly with increasing number of nodes. An asynchronous parallel implementation could be developed to address this problem with the added benefit of permitting high parallel efficiency on inhomogeneous clusters. Our results for the analytical and biomechanical optimization problems suggest that PSO performs best on problems with continuous rather than discrete noise. The algorithm consistently found the global minimum for the Griewank problem, even when the number of particles was low. Though the global minimum is unknown for the biomechanical problem using synthetic data with noise, multiple PSO runs consistently converged to the same solution. Both of these problems utilized continuous, sinusoidal noise functions. In contrast, PSO did not converge to the global minima for the Corana problem with its discrete noise function. Thus, for large-scale problems with multiple local minima and discrete noise, other optimization algorithms such as GA may provide better results [48].

Use of a LHS rather than uniform random sampling to generate initial points in design space may be a worthwhile PSO algorithm modification. Experimentation with our random number generator indicated that initial particle positions can at times be grouped together. This motivated our use of LHS to avoid re-sampling the same region of design space when providing initial guesses to sub-swarms. To investigate the influence of sampling method on PSO convergence rate, we performed multiple runs with the Griewank problem using uniform random sampling and a LHS with the default design variable bounds (−600 to +600) and with the bounds shifted by 200 (−400 to +800). We
found that when the bounds were shifted, convergence rate with uniform random sampling changed while it did not with a LHS. Thus, swarm behavior appears to be influenced by sampling method, and a LHS may be helpful for minimizing this sensitivity.

A secondary motivation for running the analytical test problems with different numbers of particles was to determine whether the use of sub-swarms would improve convergence. The question is whether a larger swarm where all particles communicate with each other is more efficient than multiple smaller swarms where particles communicate within each sub-swarm but not between sub-swarms. It is possible that the global best position found by a large swarm may unduly influence the motion of all particles in the swarm. Creating sub-swarms that do not communicate eliminates this possibility. In our approach, we performed the same number of fitness evaluations for each population size. Our results for both analytical test problems suggest that when a large numbers of processors are available, increasing the swarm size will increase the probability of finding a better solution. Analysis of PSO convergence rate for different numbers of particles also suggests an interesting avenue for future investigation. Passing an imaginary curve through the triangles in Figure 14 reveals that for a fixed number of fitness evaluations, convergence rate increases asymptotically with decreasing number of particles. While the solution found by a smaller number of particles may be a local minimum, the final particle positions may still identify the general region in design space where the global minimum is located. Consequently, an adaptive PSO algorithm that periodically adjusts the number of particles upward during the course of an optimization may improve convergence speed. For example, an initial run with 16 particles could be
performed for a fixed number of fitness evaluations. At the end of that phase, the final positions of those 16 particles would be kept, but 16 new particles would be added to bring the total up to 32 particles. The algorithm would continue using 32 particles until the same number of fitness evaluations was completed. The process of gradually increasing the number of particles would continue until the maximum specified swarm size (e.g. 128 particles) was analyzed. To ensure systematic sampling of the design space, a LHS would be used to generate a pool of sample points equal to the maximum number of particles and from which sub-samples would be drawn progressively at each phase of the optimization. In the scenario above with a maximum of 128 particles, the first phase with 16 particles would remove 16 sampled points from the LHS pool, the second phase another 16 points, the third phase 32 points, and the final phase the remaining 64 points.

**Conclusions**

In summary, the parallel Particle Swarm Optimization algorithm presented in this chapter exhibits excellent parallel performance as long as individual fitness evaluations require the same amount of time. For optimization problems where the time required for each fitness evaluation varies substantially, an asynchronous implementation may be needed to reduce wasted CPU cycles and maintain high parallel efficiency. When large numbers of processors are available, use of larger population sizes may result in improved convergence rates to the global solution. An adaptive PSO algorithm that increases population size incrementally may also improve algorithm convergence characteristics.
CHAPTER 5
IMPROVED GLOBAL CONVERGENCE USING MULTIPLE INDEPENDENT OPTIMIZATIONS

Overview

This chapter presents a methodology for improving the global convergence probability in large scale global optimization problems in cases where several local minima are present. The optimizer applied in this methodology is the PSO, but the strategy outlined here is applicable to any type of algorithm. The controlling idea behind this optimization approach is to utilize several independent optimizations, each using a fraction of a budget of computational resources. Although optimizations may have a limited probability of convergence individually as compared to a single optimization utilizing the full budget, it is shown that when they are combined they will have a cumulative convergence probability far in excess of the single optimization.

Since the individual limited optimizations are independent they may be executed concurrently on separate computation nodes with no interaction. This exploitation of parallelism allows us to vastly increase the probability of convergence to the global minimum while simultaneously reducing the required wall clock time if a parallel machine is used.

Introduction

If we consider the general global unconstrained optimization problem for the real valued function $f(x)$ defined on the set $x \in D$ in $\mathbb{R}^n$ one cannot state that a global solution has been found unless an exhaustive search of the set is $x \in D$ is performed.
With a finite number of function evaluations, at best we can only estimate the probability of arriving at or near the global optimum. To solve global optimization problems reliably the optimizer needs to achieve an efficient balance between sampling the entire design space and directing progressively more densely spaced sampling points towards promising regions for a more refined search [116]. Many algorithms achieve this balance, such as the deterministic DIRECT optimizer [117] or stochastic algorithms such as genetic algorithms [118], simulated annealing [119,120], clustering [121], and the particle swarm optimizer [122].

Although these population-based global optimization algorithms are fairly robust, they can be attracted, at least temporarily, towards local optima which are not global (see, for example, the Griewank problem in Figure 17).

Figure 17  Multiple local minima for Griewank analytical problem surface plot in two dimensions

This difficulty can be addressed by allowing longer optimization runs or an increased population size. Both these options often result in a decrease in the algorithm efficiency, with no guarantee that the optimizer will escape from the local optimum.
It is possible that restarting the algorithm when it gets stuck in a local minimum and allowing multiple optimization runs may be a more efficient approach. This follows from the hypothesis that several limited independent optimization runs, each with a small likelihood of finding the global optimum, may be combined in a synergistic effort which yields a vastly improved global convergence probability. This approach is routinely used for global search using multi-start local optimizers [123]. Le Riche and Haftka have also suggested the use of this approach with genetic algorithms for solving complex composite laminate optimization problems [124].

The main difficulty in the application of such a multi-run strategy is deciding when the optimizer should be stopped. The objective of this manuscript is to solve this problem by developing an efficient and robust scheme by which to allocate computational resources to individual optimizations in a set of multiple optimizations. The organization of this manuscript is as follows: First a brief description of the optimization algorithm applied in this study, the PSO algorithm is given. Next a set of analytical problems are described, along with details on calculating global convergence probabilities. After that, the multiple run methodology is outlined and a general budget strategy is presented for dividing a fixed number of fitness evaluations among multiple searches on a single processor. The use of this method on a parallel processing machine is also discussed. Then, numerical results based on the multiple run strategy for both single and multi-processor machines are reported and discussed. Finally, general conclusions about the multi-run methodology are presented.
Methodology

Analytical Test Set

The convergence behavior of the PSO algorithm was analyzed with the Griewank [108], Shekel [114] and Hartman [114] analytical problems (see Appendix A for problem definitions), each of which possess multiple local minima. Analytical test problems were used because global solutions $f^*$ are known a-priori. The known solution value allows us to ascertain if an optimization has converged to the global minimum. To estimate the probability of converging to the global optimum, we performed 1000 optimization runs for each problem, with each run limited to 500,000 fitness evaluations. These optimization runs are performed with identical parameter settings, with the exception of a different random number seed for each optimization run in order to start the population at different initial points in the design space. To evaluate the global convergence probability of the PSO algorithm as a function of population size, we solved each problem using a swarm of 10, 20, 50 and 100 particles. A standard set of parameters were used for the other algorithm parameters (Table 6).

Table 6  Particle swarm algorithm parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>Cognitive trust parameter</td>
<td>2.0</td>
</tr>
<tr>
<td>$c_2$</td>
<td>Social trust parameter</td>
<td>2.0</td>
</tr>
<tr>
<td>$w_0$</td>
<td>Initial inertia</td>
<td>1</td>
</tr>
<tr>
<td>$w_d$</td>
<td>Inertia reduction parameter</td>
<td>0.01</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>Bound on velocity fraction</td>
<td>0.5</td>
</tr>
<tr>
<td>$v_d$</td>
<td>Velocity reduction parameter</td>
<td>0.01</td>
</tr>
<tr>
<td>$d$</td>
<td>Dynamic inertia/velocity reduction delay (function evaluations)</td>
<td>200</td>
</tr>
</tbody>
</table>
We assumed that convergence to the global optimum was achieved when the fitness value was within a predetermined fixed tolerance $\varepsilon_t$ (see Table 7) of the known global optimum $f^*$.

$$f \leq f^* + \varepsilon_t$$ (5.1)

For the Shekel and Hartman problems, the tolerance ensures that the minimum corresponding to the global optimum for the problem has been found. That is, because $\varepsilon \neq 0$ the exact optimum is not obtained, but if a local optimizer is started from the PSO solution found with the given tolerance it will converge to the global optimum. For the Griewank problem, however, starting a local optimizer at the PSO solution will not guarantee convergence to the global optimum, since this noisy, shallow convex problem has several local minima grouped around the global optimum that will defeat a local optimizer.

Table 7  Problem convergence tolerances

<table>
<thead>
<tr>
<th>Problem</th>
<th>Convergence tolerance $\varepsilon_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Griewank</td>
<td>0.1</td>
</tr>
<tr>
<td>Shekel</td>
<td>0.001</td>
</tr>
<tr>
<td>Hartman</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Multiple-run Methodology

The use of multiple optimizations using a global optimizer such as a GA was first proposed by Le Riche and Haftka [124]. However, no criterion was given on the division of computational resources between the multiple optimizations, and the efficiency of the approach was not investigated. The method entails running multiple optimizations with a reduced number of fitness evaluations, either by limiting the number of algorithm iterations or reducing the population while keeping the number of iterations constant. Individually, the convergence probability of such a limited optimization may only be a
fraction of a single traditional optimization run. However, the cumulative convergence probability obtained by combining the limited runs can be significantly higher than that of the single run. Previously similar studies have been undertaken to investigate the efficiency of repeated optimizations using simple search algorithms such as pure random search, grid search, and random walk [130,131]. The use of multiple local optimizations or clustering [132] is a common practice but for some algorithms the efficiency of this approach decreases rapidly when problems with a high number of local minima are encountered [130].

For estimating the efficiency of the proposed strategy and for comparison with optimizations with increased populations/allowed iterations, we are required to calculate the probability of convergence to the global optimum for an individual optimization run, $P_i$. This convergence probability cannot be easily calculated for practical engineering problems with unknown solutions. For the set of analytical problems however, the solutions are known and a large number of optimizations of these problems can be performed at little computational cost. With some reasonable assumptions these two facts allow us to estimate the probability of convergence to the global optimum for individual optimization runs. The efficiency and exploration run considerations derived from the theoretical analytical results are equally applicable to practical engineering problems where solutions are not known a priori. The first step in calculating $P_i$ is using the convergence ratio, $C_r$, which is calculated as follows:

$$C_r = \frac{N_c}{N}$$

(5.2)

where $N_c$ is the number of globally converged optimizations and $N$ is the number of optimizations, in this case 1000. For a very large number of optimizations the probability
that any individual run converges to the global optimum approaches \( C_r \). For a finite number of runs however the standard error \( s_e \) in \( P_i \) can be quantified using:

\[
s_e = \sqrt{\frac{P_i(1-P_i)}{N}}
\]

which is an estimate of the standard deviation of \( C_r \). For example, if we obtain a convergence probability of \( P_i = 0.5 \) with \( N = 1000 \) optimizations, the standard error would be \( s_e = 0.016 \).

To obtain the combined cumulative probability of finding the global optimum by multiple independent optimizations we apply the statistical law for calculating the probability for success with repeated independent events. We denote the combined or cumulative probability of \( N \) multiple independent optimization runs converging to the solution as \( P_c \), and using the fact that the convergence events are uncorrelated then

\[
P_c = 1 - \prod_{i=1}^{N} P_i
\]

where \( P_i \) is the probability of the \( i \)th single individual optimization run converging to the global optimum. If we assume that individual optimization runs with similar parameter settings, as in the case of the following study, have equal probability of convergence we can simplify Eq. (5.4) to

\[
P_c = 1 - (1 - P_i)^N
\]

The increase in cumulative probability \( P_c \) with fixed values of \( P_i \) for increasing number of optimization runs \( N \) is illustrated in Figure 18. It must be stressed that the above relations are only valid for uncorrelated optimizations, which may not be the case when a poor quality random number generator is used to generate initial positions in the design space. Certain generators can exhibit a tendency to favor regions in the design space, biasing the search and probability of convergence to a minimum in these regions.
To verify the cumulative probability values predicted in theory with Eq. (5.5), the Monte Carlo method is applied, sampling random pairs, triplets, quintuplets etc. of optimizations in the pool of 1000 runs. For example, to estimate the experimental global convergence probability of two runs, we selected a large number of random pairs of optimizations among the 1000 runs. Applying Eq. (5.2), the number of cases in which either or both of the pairs converged $N_c$ for the limited optimization run, divided by $N$ (total number of pairs selected) will yield the experimental global convergence probability.

**Exploratory run and budgeting scheme**

Using the multiple run methodology requires a budget strategy by which to divide a fixed budget of fitness evaluations among the independent optimization runs. The budget of fitness evaluations $n_b$ is usually dictated by how much time the user is willing to allocate on a machine in order to solve a problem divided by how long a single fitness evaluation takes to execute. An exploratory optimization utilizing a fraction, $n_l$ of this
budget is required to determine the interaction between the optimizer and problem. The fitness history of this optimization is used to obtain an estimate of the number of fitness evaluations to be allocated to each run, \( n_i \). This strategy is based on the assumption that a single fitness history will be sufficient to quantify the optimizer behavior on a problem. For the set of problems it is observed that a correlation exists between the point where the fitness history levels off and the convergence history levels off (Figure 19).

![Figure 19](image)

**Figure 19** Fitness history and convergence probability \( P_c \) plots for Griewank, Hartman and Shekel problems

We hypothesize that the algorithm will converge quickly to the optimum or stall at a similar number of fitness evaluations (Figure 20). The exploratory run is stopped using a stopping criterion which monitors the rate of change of the objective fitness value as a function of the number of fitness evaluations. As soon as this rate of improvement drops below a predetermined value (i.e. the fitness value plot levels off), the exploratory optimization is stopped and the number of fitness evaluations noted as \( n_l \). The stopping criterion parameters used for obtaining the numerical results is a change of less than 0.01 in fitness value for at least 500 functions evaluations.
The remainder of the budgeted fitness evaluations is distributed among a number of $N$ independent optimizations, which may be calculated as follows

$$N = \left\lceil \frac{n_s - n_i}{n_i} \right\rceil$$

with an allowed number of fitness evaluations per run calculated by

$$n_i = \left\lceil \frac{n_s - n_i}{N} \right\rceil, \quad n_i \geq n_i$$

If a multi-processor machine is available, very high $P_c$ values may be reached using a multiple run strategy. If we take the simple case where each independent optimization run is assigned to a separate node, the multiple run approach will be constrained somewhat differently than the previous single-processor case. Rather than the number of multiple optimizations being limited by a fixed budget of fitness evaluations (which is divided equally among the set of multiple independent optimizations using Eq.(5.6)), the number of optimization runs will be defined by the number of computational nodes and the wall clock time available to the user. A similar method to that followed for a single

Figure 20  Typical Shekel fitness history plots of 20 optimizations (sampled out of 1000)
processor machine for determining algorithm/problem behavior must still be followed to
determine the optimal number of fitness evaluations for a single independent
optimization run. This exploratory run can, however, be done using a parallel
implementation of the population-based algorithm under consideration, in which
concurrent processing is achieved through functional decomposition [125].

**Bayesian convergence probability estimation**

If the amount of time and the global probability of convergence are competing
considerations a Bayesian convergence probability estimation method may be used, as
proposed by Groenwold et al. [134,135]. This criterion states that the optimization is
stopped once a certain confidence level is reached, which is that the best solution found
among all optimizations $\tilde{f}$ will be the global solution $f^*$. This probability or confidence
measure is given in [135] as

$$\text{Pr}[\tilde{f} = f^*] \geq q = 1 - \frac{(N + a)!((2N + b))!}{(2N + a)!((N + b))!}$$

(5.8)

where $q$ is the predetermined confidence level set by the user, usually 0.95. $N$ is the total
number of optimizations performed up to the time of evaluating the stopping criteria, and
$\bar{a} = a + b - 1$, $\bar{b} = b - N_c - 1$ with $a$, $b$ suitable parameters of a Beta distribution $\beta(a,b)$.

The number of optimizations among the total $N$ which yield a final value of $\tilde{f}$ is defined
as $N_c$. Values of parameter $a$ and $b$ were chosen as 1 and 5 respectively, as recommended
by Groenwold et al.[135].
Numerical Results

Multi-run Approach for Predetermined Number of Optimizations

For the three problems under consideration only a limited improvement in global convergence probability is achieved by applying the traditional approaches of increasing the number of fitness evaluations or the population size (Figure 21).

![Graph showing convergence probability vs. fitness evaluations for different population sizes.](image)

Figure 21  Shekel convergence probability for an individual optimization as a function of fitness evaluations and population size

For the Shekel problem, using larger swarm sizes and/or allowing an increased number of fitness evaluations yielded higher convergence probabilities only up to a point. Similar results were obtained for the Griewank and Hartman problem cases. On the other hand, optimizations with a small number of particles reached moderate global convergence probabilities at significantly fewer fitness evaluations than did optimizations with large swarms. This behavior was observed for all the problems in the test set (Figure 19). To exploit this behavior we replace a single optimization with several PSO runs, each with a limited population and number of iterations. These individual optimizations
utilize the same amount of resources allocated to the original single optimization (in this case the number of fitness evaluations).

To illustrate the merit of such an approach we optimize the Hartman analytical problem with and without multiple limited optimizations. We observe that for a single optimization the probability of convergence is not significantly improved by allowing more fitness evaluations, or by increasing the population size (Figure 22).

![Theoretical cumulative convergence probability $P_c$ as a function of the number of optimization runs with constant $P_i$ for the Hartman problem.](image)

**Figure 22** Theoretical cumulative convergence probability $P_c$ as a function of the number of optimization runs with constant $P_i$ for the Hartman problem. Multiple independent runs with 10 particles.

We also observe that an optimization with 10 particles quickly attains a probability of convergence of $P_i = 0.344$ after only 10,000 fitness evaluations. Using a multiple run strategy with 10 independent optimizations of 10,000 fitness evaluations yields the theoretical $P_c$ values reported in Table 8 (calculated using Eq. (5.5) with $P_i = 0.344$ and $n = 1,\ldots,10$). These values are indicated as circled data points at a sum equivalent number of fitness evaluations in Figure 22 for comparison with a single extended optimization.
The cumulative convergence probability $P_c$ using multiple optimizations is far superior to that of using a single optimization run of up to 100 particles.

Table 8  Theoretical convergence probability results for Hartman problem

<table>
<thead>
<tr>
<th>Number of runs $n$</th>
<th>Cumulative convergence probability $P_c$</th>
<th>Cumulative fitness evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.344</td>
<td>10000</td>
</tr>
<tr>
<td>2</td>
<td>0.570</td>
<td>20000</td>
</tr>
<tr>
<td>3</td>
<td>0.718</td>
<td>30000</td>
</tr>
<tr>
<td>4</td>
<td>0.815</td>
<td>40000</td>
</tr>
<tr>
<td>5</td>
<td>0.879</td>
<td>50000</td>
</tr>
<tr>
<td>6</td>
<td>0.920</td>
<td>60000</td>
</tr>
<tr>
<td>7</td>
<td>0.948</td>
<td>70000</td>
</tr>
<tr>
<td>8</td>
<td>0.966</td>
<td>80000</td>
</tr>
<tr>
<td>9</td>
<td>0.978</td>
<td>90000</td>
</tr>
<tr>
<td>10</td>
<td>0.985</td>
<td>100000</td>
</tr>
</tbody>
</table>

Multi-run Efficiency

To investigate the most efficient manner in which to divide a budget of fitness evaluations among multiple independent optimizations, we compare the efficiency of the 2, 5 10 and 12 independent optimizations of the Griewank problem (Figure 23). A budget of 200,000 fitness evaluations is allowed for optimizing this problem. This results in each optimization in the set being stopped at $n_i = 100,000, 40,000, 20,000$ and 16,500 fitness evaluations. It can be seen that using a combination of independent runs with high $P_i$ values (with a high number of the associated of fitness evaluations) or multiple runs with low $P_i$ values will not yield the same efficiency (as defined by $P_c$ per fitness evaluation).

If the predicted $P_c$ values are plotted on the same graph (Figure 23) it is observed that the two combinations of 5 and 10 optimizations yield the highest $P_c$ values for a given number of fitness evaluations. In both cases the independent runs are stopped at a number of fitness evaluations close to the point where $P_i$ levels off. The dependence of
efficiency on the choice of \( n_i \) can be explained as follows: If the independent optimization is stopped prematurely it will result in very low values of \( P_i \).

![Graph showing theoretical convergence probability \( P_c \) with sets of multiple runs for the Griewank problem](image)

**Figure 23** Theoretical convergence probability \( P_c \) with sets of multiple runs for the Griewank problem

Although a greater number of independent optimizations may then be performed within the single processor budget (Eq. (5.6)) it may still result in very poor efficiency, such as the 12 independent run case in Figure 23. If on the other hand an excessive amount of fitness evaluations are allowed for each independent run the strategy also suffer since only a reduced number of optimizations can then be performed (see the 2 independent run case in Figure 23). To maximize the efficiency of the multi-run strategy it is therefore desirable to allow only a number of fitness evaluations corresponding to the point where \( P_i \) starts leveling off.

From the above we can conclude that to obtain maximum efficiency, the individual runs should be terminated after reaching a number of fitness evaluations corresponding to
the point where $P_i$ starts leveling off. The exact $P_i$ convergence probability data, however, is not available unless a great number of optimizations are performed, and must be estimated. This estimation is done by observing where the fitness history of a single optimization starts leveling off. The impact and robustness of using a single exploratory optimization fitness history to determine $n_i$ for each optimization was investigated for all three analytical problems. The exploratory optimization was stopped using a rate of change stopping criteria, and $N$ and $n_i$ was calculated using (5.6)-(5.7). To verify that a single optimization will be sufficient to give robust results the rate of change stopping criteria was applied to the pool of 1000 optimizations for each of the three analytical test problems, and yielded minimum, maximum and median values of $n_i$ reported in Table 9.

Table 9 Minimum, maximum and median fitness evaluations when applying ratio of change stopping criteria on pool of 1,000 optimizations for Griewank, Hartman and Shekel problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Minimum $n_i$</th>
<th>Median $n_i$</th>
<th>Maximum $n_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Griewank</td>
<td>23,480</td>
<td>31,810</td>
<td>45,780</td>
</tr>
<tr>
<td>Hartman</td>
<td>14,180</td>
<td>16,440</td>
<td>23,360</td>
</tr>
<tr>
<td>Shekel</td>
<td>15,460</td>
<td>18,660</td>
<td>35,860</td>
</tr>
</tbody>
</table>

The minimum and maximum $n_i$ fitness evaluation history plots and corresponding convergence probability plots are given in Figure 24. It can be seen that applying the rate of change stopping criteria on a single run to estimate $n_i$ gives fairly robust results, with only a small variation in $P_c$ efficiency for all problems.

**Bayesian Convergence Probability Estimation**

The Bayesian budgeting scheme (Eq. (5.8)) indicates a consistently conservative estimation of the cumulative convergence probability for the Griewank (Figure 25), Hartman (Figure 26), and Shekel (Figure 27) problems. In order to show the accuracy of this method as the $P_c$ values approach 1 the probability of failure ($1 - P_c$) is used on a
logarithmic ordinate axis. The Bayesian estimation method is sensitive to the problem and type of optimization algorithm under consideration, and values of $a$ and $b$ parameters in the Beta distribution require fine tuning to obtain a more accurate estimation.

Figures 24 Theoretical convergence probability $P_c$ using information from exploratory optimizations which are stopped using a rate of change stopping condition for the Griewank, Hartman and Shekel problems. A solid line denotes the longest exploratory run in the pool of 1000 optimizations, and a dashed line denote shortest.

The parameters recommended by Groenwold et.al. [135], however, yield a consistently conservative estimation of the confidence level. The results suggest the Bayesian prediction of $P_c$ may be useful when confidence in the solution is traded off with the optimization time, on either a single or parallel processor machine.
Figure 25  Bayesian $P_c$ estimation comparison to using extrapolated and randomly sampled optimizations out of pool of 1000 runs for Griewank problem.

Figure 26  Bayesian $P_c$ estimation comparison to using extrapolated and randomly sampled optimizations out of pool of 1000 runs for Hartman problem.
Monte Carlo Convergence Probability Estimation

To verify the cumulative probability values predicted in theory with Eq. (5.5), the Monte Carlo method is applied, sampling random pairs, triplets, quintuplets etc. of optimizations in the pool of 1000 runs. For example, to estimate the experimental global convergence probability of two runs, we selected a large number of random pairs of optimizations among the 1000 runs. Applying Eq. (5.2), the number of cases in which either or both of the pairs converged $N_c$ for the limited optimization run, divided by $N$ (total number of pairs selected) will yield the experimental global convergence probability. This was done for Figure 22 to Figure 24, and presented in Appendix B.

Conclusions

For the set of large scale optimization problems evaluated with the PSO, the multi-run strategy with small PSO populations delivers higher global convergence probability than a single run with a large population and an equal number of fitness evaluations. On
both serial and parallel machines, a fraction of the allocated budget of fitness evaluations or computer time is required to evaluate the optimizer/problem behavior. This exploratory optimization is terminated using a rate of change stopping criteria. The number of fitness evaluations required by exploratory run is used to calculate the total number of runs and the remainder of the budget of evaluations is divided among them. This approach allows the strategy to utilize the computational resources efficiently, preventing premature termination or wasted fitness evaluations on already converged optimizations. Close correlation between theoretically predicted cumulative convergence probability and the experimentally sampled probability is obtained for the strategy on a single processor machine.

A Bayesian convergence probability estimation method may be used to stop a serial or parallel optimization when optimization reliability is traded off with time for optimization. This Bayesian prediction of the cumulative convergence probability is consistently conservative for all the problems tested when using parameters recommended in the literature.

Very high global convergence probabilities can be achieved in a limited time span on a massively parallel machine using the multi-run strategy, making this method a useful tool to solve difficult and computationally intensive large scale engineering problems.
CHAPTER 6
PARALLELISM BY DECOMPOSITION METHODOLOGIES

Overview

Some classes of engineering problems may be subdivided into several more tractable sub-problems by applying decomposition strategies. This process of decomposition generally involves identifying groups of variables and constraints with minimal influence on each other. The choice of which decomposition strategy to apply depends largely on the original problem structure and the interaction among variables.

This chapter details the application of one such methodology - the quasiseparable decomposition strategy- to a structural sizing problem. A structural optimization problem is used to illustrate the methodology of this strategy, and the resulting two-level optimization required to solve it. The research detailed in this chapter was also published as [138] in collaboration with R.T. Haftka and L.T. Watson.

Introduction

Several decomposition strategies have been proposed in the past in order to allow the solution of problems that would otherwise be too demanding computationally. One such method, the Quasiseparable decomposition strategy addresses a wide class of problems found in structural engineering. These problems allow themselves to be subdivided into lesser dimensional subsystems containing system level and component level variables. Interactions among these subsystems occur only through system level variables. Optimizing a several lower dimensional problems is desirable over the all-at-once approach for the reason that the number of local minima may increase exponentially
for certain classes of problems. An additional bonus is that parallel processing may also be utilized.

Throughout this chapter the terms system level and component level refer to the upper and lower level of the bi-level decomposed problem. The terms global and local optimizations are used exclusively to refer to the optimization technique used, e.g. global PSO vs. local SQP.

The method outlining the decomposition and optimization approach followed for solving the above types of problems, the Quasiseparable Decomposition strategy, was recently proposed by Haftka and Watson [136]. This approach introduces the concept of a "budget" of objective function leeway assigned to each of the decomposed component subsystems. This leeway allows for a global search to be performed for each subsystem. A two-level optimization scheme is applied to the decomposed problem, with subsystem optimizations at the lower level, and coordination achieved through an upper-level optimization. Each of the subsystems is independently optimized by adjusting subsystem variables, while constrained within its allowed budget, by maximizing the constraint margins. The upper-level optimizer coordinates the subsystem optimizations by adjusting the system variables and assigned budgets.

It is formally proven in [136] that a decomposition of the system using this approach will not induce spurious local minima. Previously this strategy was applied to a portal beam example with both continuous [137] and discrete [138] variables.

The numerical example in this work consists of maximizing the deflection of an end loaded stepped cantilever beam Figure 28. This problem has its roots in structural applications where large deformations without plastic strain are desirable, for example
MEMS micro actuation devices. In this example the heights and widths and wall thicknesses of 5 sections of a stepped cantilever beam are optimized subject to aspect ratio and stress constraints in order to attain maximum tip deflection. Due to the nature of the problem there are several local minima present in the 20 dimensional design space.

This problem serves to illustrate the methodology followed to decompose a structure for solution by the quasiseparable optimization strategy. It also demonstrates the reduced search effort required to obtain the solution by the quasiseparable method as compared to the traditional all-at-once approach where all variables are optimized simultaneously.

**Quasiseparable Decomposition Theory**

The method considers the special case where a system problem may be decomposed with the objective and constraints separable into a function containing only global variables $s$ and one or more global and local variables $l^{(i)}$ in the form

$$ f(s, l) = \min_{s,l} \left[ f_0(s) + f_i(s, l^{(i)}) \right] $$

subject to

$$ g^{(0)}(s) \leq 0 $$
$$ g^{(i)}(s, l^{(i)}) \leq 0, \quad i = 1, \ldots, N $$

(6.1) (6.2)

In terms of global optimization the objective functions $f_i(s, l^{(i)})$ should have as much leeway as possible. This can be done by introducing a budget $b_i \in E$ for each objective function $f_i(s, l^{(i)})$ and attacking the subsystem constraints $g^{(i)}(s, l^{(i)}) \leq 0$ by maximizing the constraint margin $-\mu_i$ for each subsystem. This decomposition strategy can then be formulated as follows:

$$ f(s, l) = \min_{s,l} f_0(s) + \sum_{i=1}^{N} b_i $$

(6.3)
subject to

\[ g^{(0)}(s) \leq 0 \]

\[ \mu_i(s, b) \leq 0, \quad i = 1, \ldots, N \quad (6.4) \]

where \( \mu_i(s, b) \) is the (global) solution to the \( i \)th lower-level problem given by

\[
\min_{\mu_i} \mu_i
\]

subject to

\[
\max_{1 \leq j \leq m_h} g^{(i)}_j(s, l^{(i)}) - \mu_i \leq 0
\]

\[
f_i(s, l^{(i)}) - b_i - \mu_i(s, b) \leq 0, \quad i = 1, \ldots, N \quad (6.6)
\]

The above strategy is only appropriate to minimization however. Since the optimization in the following example is targeted towards maximization, with the standard form

\[
f(s, l) = \max_{s_d} f_0(s) + f_i(s, l^{(i)}) \quad (6.7)
\]

subject to

\[
g^{(0)}(s) \leq 0
\]

\[
g^{(i)}(s, l^{(i)}) \leq 0, \quad i = 1, \ldots, N \quad (6.8)
\]

We need to recast the above decomposition and optimization method into a strategy where a set of performance measures \( d_i \) is maximized, in place of the minimization of budgets \( b_i \). Again, in terms of global optimization the objective functions \( f_i(s, l^{(i)}) \) should have as much leeway as possible. In place of using minimization of budgets we establish performance targets \( d_i \in E \) for each objective function \( f_i(s, l^{(i)}) \). Similar to the minimization strategy we attack the subsystem constraints \( g^{(i)}(s, l^{(i)}) \leq 0 \) by maximizing the constraint margin \( \mu_i \) for each subsystem. This yields the following

\[
f(s, l) = \max_{s_d} f_0(s) + \sum_{i=1}^N d_i \quad (6.9)
\]

subject to
\[
\begin{align*}
\psi^{(0)}(s) & \leq 0 \\
-\mu_i(s,d) & \leq 0, \quad i = 1, \ldots, N
\end{align*}
\] (6.10)

where \( \mu_i(s,d) \) is the (global) solution to the \( i \)th lower level problem given by
\[
\max_{\mu_i} \mu_i
\] (6.11)

which may be computed by selecting the maximum constraint margin and maximizing
\[
\max_{1 \leq j \leq m_i} g_j^{(i)}(s, f^{(i)}) + \mu_i \leq 0
\] (6.12)

\[
d_i - f_i(s, f^{(i)}) + \mu_i(s, d) \leq 0, \quad i = 1, \ldots, N
\]

alternatively formulated as
\[
\mu_i = \min \left( \min_{1 \leq j \leq m_i} \left( g_j^{(i)}(s, f^{(i)}) \right), \left( f_i(s, f^{(i)}) - d_i \right) \right), \quad i = 1, \ldots, N
\] (6.13)

**Stepped Hollow Cantilever Beam Example**

For an illustration of the application of the quasiseparable decomposition method to a structural problem we consider a hollow stepped cantilever beam (see Figure 28) consisting of 5 sections, each section defined by four design variables, width \((w)\), height \((h)\), top and bottom wall thickness \((t_b)\) and left and right wall thickness \((t_w)\) (see Figure 29). A load \( P \) with \( y \) component \( P_y \) and \( z \) component \( P_z \) is applied to the tip of the beam.

The tip displacement of this prismatic cantilever beam is maximized subject to a material stress constraint in addition to aspect ratio constraints. Equations describing tip displacement contributions \( \delta_y \) and \( \delta_z \) for each section \( i \) can be obtained using Castigliano’s method.

\[
\delta_y = \int_0^L \frac{M}{EI} \frac{\partial M}{\partial P} dx
\]

\[
= \int_0^L \frac{P_y x_5^2}{EI_{z_5}} dx + \int_0^L \frac{P_y (x_4 + l_5)^2}{EI_{z_4}} dx + \int_0^L \frac{P_y (x_3 + l_5 + l_4)^2}{EI_{z_3}} dx
\]

\[
+ \int_0^L \frac{P_y (x_2 + l_5 + l_4 + l_3)^2}{EI_{z_2}} dx + \int_0^L \frac{P_y (x_1 + l_5 + l_4 + l_3 + l_2)^2}{EI_{z_1}} dx
\] (6.14)

where \( E \) is the Young’s modulus and \( I_{z_j} \) through \( I_{z_5} \) are the moment of inertia’s about the neutral \( z \)-axis of the particular beam section under consideration.
Figure 28  Stepped hollow cantilever beam

Figure 29  Dimensional parameters of each cross section
If we choose \( l_1 = l_2 = l_3 = l_4 = l_5 = 1 \text{m} \), the above becomes
\[
\delta_y = \frac{P_y}{3EI_z} + \frac{7P_y}{3EI_z} + \frac{19P_y}{3EI_z} + \frac{37P_y}{3EI_z} + \frac{61P_y}{3EI_z} \\
= \delta_{y5} + \delta_{y4} + \delta_{y3} + \delta_{y2} + \delta_{y1}
\] (6.15)

Similarly the displacement in the \( z \)-direction is
\[
\delta_z = \frac{P_z}{3EI_{z5}} + \frac{7P_z}{3EI_{z4}} + \frac{19P_z}{3EI_{z3}} + \frac{37P_z}{3EI_{z2}} + \frac{61P_z}{3EI_{z1}} \\
= \delta_{z5} + \delta_{z4} + \delta_{z3} + \delta_{z2} + \delta_{z1}
\] (6.16)

![Projected displacement in direction \( \theta \)](image)

Figure 30  Projected displacement in direction \( \theta \)

The tip deflection maximization is subject to the stress constraints for each beam section \( i \) (with the allowable stress reduced by safety factor \( SF \)). The stresses are calculated at the root of each beam segment, at a maximum distance from the bending axis (four corners of each root cross section) using the Euler-Bernoulli beam equations.

\[
\sigma_i - \frac{\sigma_{allow}}{SF} \leq 0
\]

\[
\left( \frac{M_Q h_i}{2I_z} \right)_i + \left( \frac{M_R W_i}{2I_y} \right)_i - \frac{\sigma_{allow}}{SF} \leq 0
\] (6.17)

with
\[
I_{yi} = 2 \left[ \frac{t_{hi} (w_i - 2t_{wi})}{12} + \frac{h_i t_{wi}^3}{12} + t_{wi} h_i \left( \frac{w_i - t_{wi}}{2} \right)^2 \right]
\]

\[
I_{zi} = 2 \left[ \frac{(w_i - 2t_{wi}) h_i^3}{12} + \frac{t_{wi} h_i^3}{12} + t_{hi} (w_i - 2t_{wi}) \left( \frac{h_i-t_{hi}}{2} \right)^2 \right]
\] (6.18)

In addition to the stress constraints the aspect ratio of each beam segment is limited to...
\[ 0.2 \leq \frac{w_i}{h_i} \leq 5 \]  
(6.19)

The hollow cavity along the axial direction and lower bounds on wall thicknesses are accommodated through the following geometric constraints

\[ 0.01 \leq t_{hi} \leq \frac{h_i}{4}, \]
\[ 0.01 \leq t_{wi} \leq \frac{w_i}{4} \]
(6.20)

The above constraints are then normalized,

\[ g_1 = \left[ \sigma_i \left/ \left( \frac{\sigma_{allow}}{SF} \right) \right. \right] - 1 \leq 0 \]
\[ g_2 = \frac{4t_{hi}}{h_i} - 1 \leq 0 \]
\[ g_3 = \frac{4t_{wi}}{w_i} - 1 \leq 0 \]
\[ g_4 = 1 - \frac{t_{hi}}{0.01} \leq 0 \]  
(6.21)
\[ g_5 = 1 - \frac{t_{wi}}{0.01} \leq 0 \]
\[ g_6 = \left[ \frac{w_i}{h_i} \right] / 5 - 1 \leq 0 \]
\[ g_7 = \left[ \frac{h_i}{w_i} \right] / 5 - 1 \leq 0 \]

This yields seven constraints per section, for a total of 35 constraints. The beam material properties and applied load is given in Table 10

<table>
<thead>
<tr>
<th>Material property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Young's Modulus $E$</td>
<td>72e9 Pa</td>
</tr>
<tr>
<td>Safety factor $SF$</td>
<td>3</td>
</tr>
<tr>
<td>Allowable stress $\sigma_{allow}$</td>
<td>505e6 Pa</td>
</tr>
<tr>
<td>$l_1, l_2, l_3, l_4, l_5$</td>
<td>1m.</td>
</tr>
<tr>
<td>$P_y$</td>
<td>3535.5 N</td>
</tr>
<tr>
<td>$P_z$</td>
<td>3535.5 N</td>
</tr>
</tbody>
</table>
**Stepped hollow beam optimization.**

For comparative purposes the above optimization problem is first solved using the traditional “all-at-once” fashion (single level optimization) with the entire set of 20 design parameters as input variables. The system is then decomposed using the quasiseparable strategy, and optimized as 5 sets of 4 dimensional sub-problems. For both approaches the tip displacement is maximized subject to the stated stress and geometric constraints. Using the principle of superposition and considering contributions by individual sections to the tip displacements in the \(z\) and \(y\) directions separately, we can formulate the tip displacement \(\delta_\theta\) projected in a specified direction \(\theta\) (see Figure 30) as follows:

\[
\delta_\theta = \sum_{i=1}^{5} \left( \delta_{yi} \cos \theta + \delta_{zi} \sin \theta \right)
\]

(6.22)

where \(\delta_{yi}\) and \(\delta_{zi}\) are the section tip displacement contributions in the \(y\) and \(z\) directions respectively. The maximum tip displacement may be calculated using Eq. (6.22)

\[
\max_{w_1,b_1,t_{1s},t_{1f},w_2,b_2,...,w_5,b_5,t_{5s},t_{5f}} \delta = \sum_{i=1}^{5} \delta_{yi} \cos \theta + \sum_{i=1}^{5} \delta_{zi} \sin \theta
\]

(6.23)

The maximization of the tip deflection (Eq. (6.22)) can then be reformulated as follows

\[
\max_{w_1,b_1,t_{1s},t_{1f},w_2,b_2,...,w_5,b_5,t_{5s},t_{5f}} \delta = \sum_{i=1}^{5} \delta_i
\]

(6.24)

with

\[
\delta_i = \delta_{yi} \cos \theta + \delta_{zi} \sin \theta
\]

(6.25)

There are 2 local optima for each beam section when considering a solid beam cross section. This is explained by the fact that either the beam width or height will be the dominantly thin optimized dimension, which is primarily constrained by the yield stress and aspect ratio constraints (Eq. (6.19)). The maximum tip deflection may therefore be
achieved in two equally possible directions, if \( \theta \) is chosen as \( \theta = \frac{\pi}{4} \). As an example, these two possible local minima are shown in Figure 31 for a solid cross section beam.

![Figure 31](image)

Figure 31 Tip deflection contour plot as a function of beam section 5 with height \( h \), and width \( w \) with yield stress and aspect ratio constraints indicated by dashed and dash dotted lines respectively. Crosses indicate the two solutions (one dominated by bending in \( y \)-axis, and the other in the \( z \)-axis) for maximum tip displacement.

This yields a total of \( 2^5 = 32 \) possible local minima for a solid beam, which are the possible combinations of tall thin sections (dominant bending in \( z \)-direction) and flat thin sections (dominant bending in \( y \)-direction). For the purpose of clearly identifying the global minimum in this work, we choose \( \theta = \frac{\pi}{3} \), which biases the tip deflection in the \( y \) direction. With the additional two design variables \( t_w \) and \( t_h \) for each section, used to define the hollow beam, several additional local minima are induced in the system, but the global minimum remains biased in the \( z \) direction. The global solution to the hollow beam and the associated design vectors and normalized constraint values are given in Table 11. The total tip deflection of this optimal design is 0.755m. It can be seen in Table
11 that the constraints $g_4$ and $g_5$, which require the hole to be at least 10mm square are not active for all but the last section, and then only in $g_5$.

Table 11  Stepped hollow beam global optimum

<table>
<thead>
<tr>
<th>Section $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta_{i,t}$ (Tip deflection contribution m.)</td>
<td>0.2579</td>
<td>0.2107</td>
<td>0.1587</td>
<td>0.1004</td>
<td>0.0267</td>
</tr>
<tr>
<td>$w$ (width, m.)</td>
<td>0.2722</td>
<td>0.2527</td>
<td>0.2296</td>
<td>0.2005</td>
<td>0.1136</td>
</tr>
<tr>
<td>$h$ (height m.)</td>
<td>0.0544</td>
<td>0.0505</td>
<td>0.0459</td>
<td>0.0401</td>
<td>0.0400</td>
</tr>
<tr>
<td>$t_{wi}$ (thickness, m.)</td>
<td>0.0680</td>
<td>0.0632</td>
<td>0.0574</td>
<td>0.0506</td>
<td>0.0284</td>
</tr>
<tr>
<td>$t_{hi}$ (thickness, m.)</td>
<td>0.0136</td>
<td>0.0126</td>
<td>0.0115</td>
<td>0.0100</td>
<td>0.0100</td>
</tr>
<tr>
<td>constraint $g_{1*}$</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>constraint $g_{2*}$</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>constraint $g_{3*}$</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>constraint $g_{4*}$</td>
<td>-5.8044</td>
<td>-5.3167</td>
<td>-4.7391</td>
<td>-4.0136</td>
<td>-1.8399</td>
</tr>
<tr>
<td>constraint $g_{5*}$</td>
<td>-0.3609</td>
<td>-0.2633</td>
<td>-0.1478</td>
<td>-0.0027</td>
<td>0.0000</td>
</tr>
<tr>
<td>constraint $g_{6*}$</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>-0.4320</td>
</tr>
<tr>
<td>constraint $g_{7*}$</td>
<td>-0.9600</td>
<td>-0.9600</td>
<td>-0.9600</td>
<td>-0.9600</td>
<td>-0.9296</td>
</tr>
</tbody>
</table>

* Constraints defined in Eq. (6.21)

**Quasiseparable Optimization Approach**

By reformulating the equations governing the tip displacement (Eq. (6.22)) into the summation of individual contributions (Eq (6.24)) we observed that they are of the quasiseparable form (Eq.(6.9)). This allows us to optimize the problem using the decomposed formulation in Eq. (6.9)-(6.13).

$$f(\theta, w_1, h_1, t_{w1}, t_{h1}, w_2, h_2, \ldots, t_{w5}, t_{h5}) = \max_{w_1, \theta, h_1, t_{w1}, t_{h1}, w_2, h_2, \ldots, t_{w5}, t_{h5}} \left[ f_0(\theta) + \sum_{i=1}^{5} d_i \right]$$  \( (6.26) \)

where the $d_i$'s are the target displacement contributions by each beam section to the total tip displacement. There is no global variable contribution ($f_0$) in this example thus (Eq (6.26)) can be written simply as

$$f(\theta, w_1, h_1, t_{w1}, t_{h1}, w_2, h_2, \ldots, t_{w5}, t_{h5}) = \max_{w_1, \theta, h_1, t_{w1}, t_{h1}, w_2, h_2, \ldots, t_{w5}, t_{h5}} \left[ \sum_{i=1}^{5} d_i \right]$$  \( (6.27) \)

subject to

$$-\mu_i(\theta, d_i) \leq 0, \quad i = 1, \ldots, 5$$  \( (6.28) \)

where $\mu_i$ is the (global) solution to the $i$th beam section optimization
which is obtained by selecting the maximum of the constraints defined in (6.21)

\[
\mu_i = \max \left( \max_{\theta, w_i, h_i} \left( d_i - f_i \left( \theta, w_i, h_i, t_{wi}, t_{hi} \right) \right) \right)
\]

Using the quasiseparable formulation of the optimization problem allows us to independently and concurrently optimize the sub-problems (sections) on a lower level. The solutions obtained in the lower level optimization are then coordinated in the upper level (Eq. (6.27)) optimization of the budgets in order to obtain the maximum tip deflection. The system level and component level optimization interactions are illustrated in Figure 32.
Results

All-at-once Approach

For the all-at-once approach an optimization was performed with the Matlab `fmincon` function, an implementation of the sequential quadratic programming algorithm. using 20 design variables and 30 constraints. The optimization converged to the total tip deflection and sectional tip deflection contributions reported in Table 12. Although all stress and geometric constraints were satisfied it can be seen that the tip deflection did not reach the optimal value of 0.7545. Repeated optimization using random starting points with the Matlab `fmincon` optimizer revealed that several such local optima exist. This is illustrated in Figure 33, which was generated by sorting the solutions of the 1000 optimizations in an ascending order and then plotting.

![Graph](image)

Figure 33  Results for 1000 all-at-once optimizations. Tip deflection values sorted ascending and plotted as a function of optimizations. Flat areas in the graph represent optimizations which converged to the same local minimum.
It can be seen that the SQP has severe difficulty obtaining the global minimum due to the presence of several local minima in the design space. The tip deflection contributions of the median optimum indicated in Figure 33 are reported in Table 12. Only a single optimization out of the 1000 converged to the global optimum.

<table>
<thead>
<tr>
<th>Table 12</th>
<th>All-at-once approach median solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Optimal tip deflection contributions</td>
</tr>
<tr>
<td>Section 1</td>
<td>0.2579</td>
</tr>
<tr>
<td>Section 2</td>
<td>0.2107</td>
</tr>
<tr>
<td>Section 3</td>
<td>0.1588</td>
</tr>
<tr>
<td>Section 4</td>
<td>0.1004</td>
</tr>
<tr>
<td>Section 5</td>
<td>0.0267</td>
</tr>
<tr>
<td>Total tip deflection</td>
<td>0.7545</td>
</tr>
</tbody>
</table>

**Hybrid all-at-once Approach**

To attempt to improve over the results obtained in the previous section (fmincon applied in the all-at-once optimization) a hybrid approach is applied to the problem. This approach used a population based search method, the PSO, coupled with the fmincon function. The solution vector obtained by the PSO is used as a starting point for the fmincon function. Due to the increase in required computational effort to solve the problem only 100 optimizations are performed. The median optimization result is given in Table 13, and the ascending fitness value plot is given in Figure 34. A marked increase in optimizations converging to the global optimum is observed when using the hybrid approach, as compared to fmincon only. The number of optimizations which converged to the global optimum however, is still less than 7% of the total, and applying this strategy comes at significant additional computational cost. Despite the improvement in the average tip deflection this method did not obtain the global optimum of 0.7545, the maximum tip deflection attained was only 0.7278 (upper plateau in Figure 34).
The quasiseparable optimization strategy is applied to the tip deflection problem by applying the Matlab *fmincon* optimizer to both the upper and lower level optimization. When optimizing the sub-problem using the *fmincon* function we observed that a substantial fraction of optimizations converged to the global optimum for that section. Figure 35. The other plateau indicate convergence to the less favorable optimum (as
previously indicated in Figure 31) and has the exact same cross section design configuration, but the section is rotated 90 degrees, which favors a displacement in the z-direction. A total of 140 runs out of the 1000 converged to the global optimum in this component optimization. This fraction of converged optimizations correspond to a 0.14 convergence probability, and led us to apply the multiple run strategy proposed in Chapter 5. The local optimizer SQP in the function \textit{fmincon} is applied repeatedly to the sub optimization, for a total of 50 optimizations. This repeated use of \textit{fmincon} proved to be far more computationally efficient at the lower level optimization compared to using a hybrid method such as PSO-\textit{fmincon}. Applying Eq (5.5) with the assumption that all runs are independent yields a failure probability of approximately $5.31e^{-4}$ for each multiple run, and a combined 0.27% for all sections during the system level optimization. This results in a 7.7% probability of failure for a problem optimization with 30 upper level iterations (typical required number of iterations for this problem).

The hybrid method was also applied to the lower level optimization sub-problem and yielded the true optimum reliably, but took considerable time because of the high number of fitness evaluations required by the PSO, the effect of which are exacerbated by multiple iterations by the upper level optimizer. From Figure 36 it can be seen that the constraint margins are reduced to at least $10^{-10}$ after only 5 iterations of the upper level optimizer.

**Approximation of Constraint Margins**

Because of the expense of repeated lower level optimizations it may be advantageous to create a surrogate approximation for calculating constraint margin values for each beam section, similar to an approach followed by Liu et. al. [139]. This surrogate serves as a simple lookup of the response maximized constraint as a function of the
section target tip displacement value at the component level optimization. For each beam section a surrogate function is created by sampling a 100 target tip deflection points and fitting a one dimensional spline to the set of constraint margin response values.

![Graph showing repeated optimizations of section 1 sub-problem using fmincon function.](image)

Figure 35  Repeated optimizations of section 1 sub-problem using \textit{fmincon} function.

Table 14  Quasiseparable optimization result

<table>
<thead>
<tr>
<th>Section</th>
<th>Optimal tip deflection contributions</th>
<th>Quasiseparable tip deflection contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 1 target</td>
<td>0.2579</td>
<td>0.2579</td>
</tr>
<tr>
<td>Section 2 target</td>
<td>0.2107</td>
<td>0.2107</td>
</tr>
<tr>
<td>Section 3 target</td>
<td>0.1588</td>
<td>0.1588</td>
</tr>
<tr>
<td>Section 4 target</td>
<td>0.1004</td>
<td>0.1004</td>
</tr>
<tr>
<td>Section 5 target</td>
<td>0.0267</td>
<td>0.0267</td>
</tr>
<tr>
<td>Total tip deflection</td>
<td>0.7545</td>
<td>0.7545</td>
</tr>
</tbody>
</table>

This reduces the cost of obtaining margin values to a simple interpolation calculation, in place of a global optimization. The reduced computational cost at the lower level allows the use of robust but expensive optimizers such as the PSO algorithm at the upper level.
The exact optimum design variables are all found to within $10^{-6}$ m tolerances. The solution is also illustrated as a sequence of cross sections in Figure 38.

**Figure 36** Summed budget value and constraint margins for individual sections.

Global optimum, $\delta_1 = 0.258$

Local optimum, $\delta_1 = 0.156$

**Figure 37** Global and local optimum in section 1 sub-optimization. Scale is 0.1:1

Numerical results obtained by using the PSO with the surrogate approach on this problem are presented in Figure 39 and Figure 40. From these figures it can be observed
that the target tip displacements for all sections reach steady values after approximately 4000 function evaluations at the upper level.

![Decomposed cross section solution](image)

**Figure 38** Decomposed cross section solution. Scale is 0.1:1

The search enters a refined stage where the constraint margins are continued to be adjusted until about 8000 fitness evaluations. The final tip displacement contributions at 10000 fitness evaluations are reported in Table 15. Values in Table 15 indicate that the tip deflection contributions are all slightly overestimated, due to the use of a penalty method in the upper level optimization, required for the accommodation of constraints.

<table>
<thead>
<tr>
<th>Section</th>
<th>Optimal tip deflection contributions</th>
<th>Quasiseparable tip deflection contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 1 target</td>
<td>0.2579</td>
<td>0.2580</td>
</tr>
<tr>
<td>Section 2 target</td>
<td>0.2107</td>
<td>0.2107</td>
</tr>
<tr>
<td>Section 3 target</td>
<td>0.1588</td>
<td>0.1588</td>
</tr>
<tr>
<td>Section 4 target</td>
<td>0.1004</td>
<td>0.1005</td>
</tr>
<tr>
<td>Section 5 target</td>
<td>0.0267</td>
<td>0.0268</td>
</tr>
<tr>
<td>Total tip deflection</td>
<td>0.7545</td>
<td>0.7547</td>
</tr>
</tbody>
</table>

**Table 15** Surrogate lower level approximation optimization results

**Discussion**

Repeated application of the all-at-once approach obtained only a single solution out of the 1000 optimizations which converged to the global optimum, giving a probability of convergence of 0.1%. The quasiseparable optimization strategy combined with a multi-
run approach at the component level yields a probability of convergence of 92.3%. The quasiseparable strategy requires, (for a typical optimization of this problem) 7750 component level optimizations, but these are of reduced dimensionality (and computational effort), when compared to the original problem formulation.

![Figure 39 Target tip deflection value histories as a function of upper-level fitness evaluations](image)

Although the component level optimizations are of reduced effort they are repeatedly performed for optimization iterations of the upper level. It may therefore be advantageous for some types of problems to approximate the response margin calculated by the lower level optimization using a surrogate model. For this particular structural problem a very simple surrogate model was constructed, a one-dimensional spline for each section of the beam.
Figure 40  Constraint margin value histories as a function of upper-level function evaluations
These models, in place of the component level optimizations maintain a high level of reliability, and deliver a reasonably accurate solution. The deviation for the solution obtained using the surrogate approach (Table 15) was caused by the penalty multiplier in the constraint handling scheme being set too low, and not by the surrogate model itself. This was proven by using set of increasingly higher penalty multipliers which improved the solution to within the same accuracy as the surrogate model (which was calculated to within 10e-6). Higher accuracies however, required additional computational effort.

**Conclusions**

This method shows promise in the fact that it can decompose weakly coupled structural problems and attack the sub-problems on a wider work front, using parallel processing and a two-level optimization scheme. The results obtained for this approach show that the strategy can recover the same solution as for the original problem formulation. It is also shown that the strategy transfers the global search effort to the lower-level component optimization, which is of lower dimensionality, thereby making the problem more tractable to the optimizer. A vastly improved probability of obtaining the global optimum is made possible by utilizing the multi-run approach presented in Chapter 5 at the component-level global optimization.

To further reduce computational effort it is demonstrated that for some problem types the component level optimizations may be replaced with a surrogate model. This approach reliably found the global optimum to the example problem with high accuracy.

The quasiseparable decomposition and optimization approach presented in this chapter is shown to be a robust and efficient means of optimizing structural problems.
CHAPTER 7
CONCLUSIONS

The approaches investigated in this dissertation yield significantly increased throughput in global optimization by means of exploiting parallelism at different levels. This allows the user to solve more complex and computationally demanding problems than previously possible by fully utilizing parallel resources. These strategies are increasingly relevant because of the rapid development of parallel processing infrastructures, and its ever more common availability.

Parallelism by Exploitation of Optimization Algorithm Structure

The PSO algorithm applied in the concurrent processing strategies outlined in Chapters 3-5 is a useful candidate global optimization method when considering continuous variable problems. The PSO also has the advantage of insensitivity to the scale of the design variables. The technique of exploiting parallelism in this algorithm presented in Chapter 4 is also applicable to other population based methods such as Genetic algorithms. The parallel Particle Swarm Optimization algorithm exhibits excellent parallel performance when individual fitness evaluations require the same amount of time. For optimization problems where the time required for each fitness evaluation varies substantially, an asynchronous implementation may be more computationally efficient.

Parallelism through Multiple Independent Optimizations

The decomposition and multiple independent optimization methods presented in Chapter 5 may be applied to problems using any optimization method appropriate to the
type of problem. For the set of large scale optimization problems evaluated with the PSO, the multi-run strategy with small PSO populations delivers higher global convergence probability than a single run with a large population and an equal number of fitness evaluations. On both serial and parallel machines, a fraction of the allocated budget of fitness evaluations or computer time is required to evaluate the optimizer/problem behavior. This fraction of the budget is used to calculate the total number of runs which will yield the most efficient distribution of the remainder of the budget. A Bayesian convergence estimation which is updated as the optimizations complete may be used to stop a serial or parallel optimization when the available time is traded off with an confidence measure. Very high global convergence probabilities can be achieved in a limited time span on a massively parallel machine using the multi-run strategy, making this method a useful tool to solve difficult and computationally intensive large scale engineering problems.

**Parallelism through Concurrent Optimization of Decomposed Problems**

The application of the quasiseparable decomposition method yields an efficient manner of distributing the global search effort required by a problem into several reduced and manageable portions. These decomposed elements of the problems may be optimized concurrently, allowing faster optimization times, as compared to an all at once solution of the problem. The use of multiple independent optimizations at the component level optimizations also improved the reliability and efficiency of the strategy, and demonstrates that two of the methods of parallelism may be effectively combined.

**Future Directions**

The multiple independent optimization run methodology should be further refined by studying multiple stopping criteria that may be used to terminate the exploratory
optimization run. The Bayesian global convergence estimation may also be improved by selecting different distributions than the Beta distribution used in this work, or by performing a parameter sensitivity study on $\alpha$ and $\beta$, which may yield a more robust or accurate estimation.

Summary

All strategies presented in this dissertation yield improved parallelism at various aspects of structural optimization. These strategies may be effectively combined as shown in the example presented in Chapter 6, obtaining increased levels of efficiency, robustness, utilization of parallel resources in the optimization of large scale problems. This will allow the use of large numbers of processors to solve optimization problems, further increasing throughput.
APPENDIX A
ANALYTICAL TEST PROBLEM SET

Griewank

Objective function:

\[ f(x) = \sum_{i=1}^{n} \frac{x_i^2}{d} - \prod_{i=1}^{n} \cos \left( \frac{x_i}{\sqrt{i}} \right) + 1 \]  

(1)

with \( n = 10 \) and \( d = 4000 \)

Search domain \( D = \{ (x_1, x_2, \ldots, x_{10}) \in \mathbb{R}^{10} : -600 \leq x_i \leq 600, i = 1, 2, \ldots, 10 \} \)

Solution: \( x^* = (0.0, 0.0, \ldots, 0.0) \), \( f^* = 0.0 \)

Hartman 6

Objective function:

\[ f(x) = -\sum_{i=1}^{m} c_i \exp \left( -\sum_{j=1}^{n} a_{ij} (x_j - p_{ij})^2 \right) \]  

(2)

Search domain \( D = \{ (x_1, \ldots, x_n) \in \mathbb{R}^{n} : 0 \leq x_i \leq 1, i = 1, \ldots, n \} \)

Solution (with \( m = 4 \)):

\( x^* = (0.2017, 0.1500, 0.4769, 0.2753, 0.3117, 0.6573, 3.322368) \), \( f^* = -3.322368 \)

See Table 16 for values of \( a_{ij}, c_i \), and \( p_{ij} \).
Shekel 10

Objective function:

\[ f(x) = \sum_{i=1}^{m} \frac{1}{(x - a_i)^T (x - a_i) + c_i} \]  

(3)

Search domain \( D = \{ x_i \in R^4 : 0 \leq x_i \leq 10, i = 1, \ldots, 4 \} \)

Solution (with \( m = 10 \)):

\[ x^* = (4.00074671, 4.00059326, 3.99966290, 3.99950981), \quad f^* = -10.536410 \]

See Table 17 for values of \( a_{ij} \) and \( c_i \).

Table 16 Hartman problem constants

<table>
<thead>
<tr>
<th>( a_{ij} )</th>
<th>( c_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0</td>
<td>3.0</td>
</tr>
<tr>
<td>0.05</td>
<td>10.0</td>
</tr>
<tr>
<td>3.0</td>
<td>3.5</td>
</tr>
<tr>
<td>17.0</td>
<td>8.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( p_{ij} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1312</td>
</tr>
<tr>
<td>0.2329</td>
</tr>
<tr>
<td>0.2348</td>
</tr>
<tr>
<td>0.4047</td>
</tr>
</tbody>
</table>
Table 17  Shekel problem constants

<table>
<thead>
<tr>
<th>m</th>
<th>$a_i$</th>
<th>$c_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.0</td>
<td>4.0</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>8.0</td>
<td>8.0</td>
</tr>
<tr>
<td>4</td>
<td>6.0</td>
<td>6.0</td>
</tr>
<tr>
<td>5</td>
<td>3.0</td>
<td>7.0</td>
</tr>
<tr>
<td>6</td>
<td>2.0</td>
<td>9.0</td>
</tr>
<tr>
<td>7</td>
<td>5.0</td>
<td>5.0</td>
</tr>
<tr>
<td>8</td>
<td>8.0</td>
<td>1.0</td>
</tr>
<tr>
<td>9</td>
<td>6.0</td>
<td>2.0</td>
</tr>
<tr>
<td>10</td>
<td>7.0</td>
<td>3.6</td>
</tr>
</tbody>
</table>
Figure 41 Predicted and Monte Carlo sampled convergence probability $P_c$ for 5 independent optimization runs for the Griewank problem. Each optimization run is limited to 40,000 fitness evaluations with 20 particles.
Figure 42  Predicted and Monte Carlo sampled convergence probability $P_c$ for 12 independent optimization runs for the Griewank problem. Each optimization run is limited to 16,000 fitness evaluations with 20 particles.

Figure 43  Monte Carlo sampled convergence probability $P_c$ with sets of multiple runs for the Griewank problem.
Figure 44 Monte Carlo sampled convergence probability $P_c$ using information from exploratory optimizations stopped using a rate of change stopping condition for the Griewank, Hartman and Shekel problems. A solid line denotes the longest exploratory run in the pool of 1000 optimizations, and a dashed line denote shortest.
Figure 45  Bayesian $P_c$ comparison for Griewank, Hartman and Shekel problem.
LIST OF REFERENCES


BIOGRAPHICAL SKETCH

Jaco Francois Schutte was born in Pretoria, South Africa, on 12 November 1975. He attended the University of Pretoria where he received both bachelor’s (1999) and master’s (2001) degrees in mechanical engineering. He also received the Sasol award for best master’s student from the Faculty of Engineering at the University of Pretoria.

Mr. Schutte continued his graduate studies at the University of Florida in Gainesville, Florida, in the pursuit of a doctoral degree.